Joonas Harjukallio

# Web-pohjainen varastonhallintasovellus

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

16.5.2013

Metropolia

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Joonas Harjukallio<br>Web-pohjainen varastonhallintasovellus<br><br>43 sivua + 5 liitettä<br>16.5.2013 |
| Tutkinto | insinööri (AMK) |
| Koulutusohjelma | tietotekniikka |
| Suuntautumisvaihtoehto | ohjelmistotekniikka |
| Ohjaaja(t) | lehtori Olli Hämäläinen |


Insinöörityön tavoitteena oli suunnitella ja toteuttaa web-pohjainen varastonhallintasovellus. Sovelluksen tarkoitus oli helpottaa yrityksen tilausten käsittelyä sekä siihen liittyvää varastonhallintaa. Varaston manuaalinen järjestyksen ylläpito aiheutti suuria kustannuksia sekä hidasti varaston toimintaa kadonneiden tuotteiden takia.

Työn tavoitteena oli suunnitella nykyaikainen varastonhallintajärjestelmä, jonka muuttaminen asiakkaan vaatimusten mukaan olisi mahdollisimman helppoa. Järjestelmä toteutettiin web-tekniikoilla, jolloin mobiili- ja työpyötälaitteet käyttävät samaa ohjelmakoodia.

Varastonhallinnan suunnittelun lisäksi sovelluksen implementoinnissa perehdyttiin MVC-arkkitehtuurillisen Ruby on Rails -web-sovelluskehyksen menetelmiin.

Sovelluksen toteuttamiseen annettiin aikaa noin kolme kuukautta. Sovellus tulisi olemaan osa tulevaisuudessa kehitettävää toiminnanohjausjärjestelmää.

Sovelluksen ensimmäisen version toteutuksessa saavutettiin asiakkaan vaatima tulos. Sovellus nopeutti varaston toimintaa sekä luotettavuutta huomattavasti. Ruby on Rails osottautui erittäin hyväksi web-sovelluskehykseksi web-pohjaisia sovelluksia kehitettäessä.



| | |
|---|---|
| Avainsanat | Ruby on Rails, varastonhallinta, REST |

| | |
|---|---|
| Author(s)<br>Title<br><br>Number of Pages<br>Date | Joonas Harjukallio<br>Web-base warehouse management system<br><br>43 pages + 5 appendices<br>15 May 2013 |
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Specialisation option | Software Engineering |
| Instructor(s) | Olli Hämäläinen, Senior Lecturer |

The purpose of the project described in this thesis was to design and implement a web-based warehouse management system. The system is supposed to simplify the order management of the company for which it was created as well as the warehouse management related to order management. Previously, analog warehouse management led to great expenses and held up warehouse response time because of lost property.

Modifying the system according to the customer's requirements should be as easy as possible. The system was implemented with web-techniques, which means that mobile and desktop devices use the same server-side code base.

In addition to warehouse management design, the MVC-architectural pattern was studied in implementing the application on the Ruby on Rails framework.

About three months was given to implement the application. The application will be part of enterprise resource planning software, which will be implemented in the future.

The first version of the application met the requirements of the customer. The application greatly sped up warehouse management and brought reliability. In conclusion, Ruby on Rails turned out to be a great tool in creating web-based applications.

| | |
|---|---|
| Keywords | Ruby on Rails, warehouse management system, REST |

## Sisällys

## Lyhenteitä ja käsitteitä

AJAX *Asynchronous JavaScript and XML*. Web-kehitys tekniikka, jolla haetaan ei-reaaliaikaisesti palvelimelta tietoa.

Bundler Ruby-sovelluksille ruby-kirjastojen välisten riippuvuuksien hallintatyökalu.

Bytecode Tavukoodi, jonka tavoite on vähentää ohjelmistotulkin ajoaikaa.

DOM Ohjelmointirajapinta, joka mahdollistaa HTML-tiedostojen rakenteen, sisällön ja tyylin dynaamisen muokkaamisen.

ERP *Enterprise resource planning*. Sovelluskokonaisuus yrityksen tuloksen parantamiseksi.

event-handle

Asynkronin takaisinkutsualirutiini, joka ottaa vastaan mm. käyttäjäsyötteitä.

Gem Paketinhallintatyökalu Ruby-kielelle.

GemFile Tiedosto, jonka avulla bundler-työkalu tietää Ruby-ohjelmistossa käytettävät ulkoisetkirjastot.

Git Jaettu lähdekoodin tarkastus ja hallintatyökalu.

Github Web-pohjainen versionhallintatyökalu, joka käyttää Git-versionhallintaa.

jQuery Plug-in

jQuery-kirjastoa laajentava sovellus.

Mass assignment

Ohjelmistonhaavoittuvuus, jossa web-pohjaisen active record-suunnittelumallin sääntöjen vastaisesti muutetaan tietokannasta löytyvää dataa.

MES *Manufacturing execution system*. Tuotannonohjaus, jonka avulla on tarkoitus parantaa yrityksen tulosta.

Metaprogramming

Ohjelmointikoodin kirjoittamista, joka muuttaa tai ohjelmointikoodia ajonaikana.

MRI *Matz's Ruby Interpreter*. Rubyn oletustulkki.

MVC *Model-View-Controller*. Ohjelmointiarkkitehtuurisuunnittelumalli, joka eriyttää käyttöliittymän ohjelma-logiikasta.

ORM *Object Relational Mapping*. Ohjelmointityökalu, jonka avulla abstrahoidaan tietokantataulut olioista.

Rails Täysimittainen web-sovelluskehys Ruby-kielelle.

Rake Ruby-pohjainen tehtävänhallintatyökalu, jonka avulla luodaan ajettavia tiedostoja automaattisesti.

REST *Representational state transfer*. Ohjelmistoarkkitehtuuri tyyli web-sovelluksille.

Ruby Yleiskäyttöinen olio-ohjelmointikieli.

Scaffold Rails komentorivikäsky, jolla luodaan runko Rails-resurssille. Käsky luo kontrollerin, mallin, näkymät, testit ja tietokantamigraatiot.

| | |
|---|---|
| SCM | *Software configuration management*. Ohjelmistokoodin muutoksen hallinta. |
| YARV | *Yet An Other Ruby VM*. Bytecode-tulkki, joka on kehitetty Ruby-kielelle. Tulkin tavoite on vähentää huomattavasti Ruby-ohjelmakoodin ajoaikaa. |
| WAMAS | *Warehouse management system*. Työssä toteutettu varastonhallintasovellus. |
| WEBrick | Yksinkertainen HTTP web-palvelin Ruby-kielelle. |

# 1 Johdanto

Tämän työn asiakas toimii autonvaraosien tukkutoimittajana. Yrityksessä on noin kymmenen vakituista työntekijää. Työntekijät ovat pääasiassa varastomiehiä, markkinoijia tai taloushallinnon työntekijöitä. Päivän aikana tulee useita tilauksia, jotka pitää toimittaa mahdollisimman nopeasti. Tuotteiden on tällöin löydyttävä helposti, ja varastosta löytyvien tuotteiden lukumäärä on oltava tiedossa. Varastoon myös saapuu tuotteita, jotka pitää pystyä varastoimaan mahdollisimman helposti.

Ennen digitaalista varastonhallintaa yritys kirjasi saapuneet sekä lähteneet tuotteet paperisten hyllykorttien avulla. Vanhanaikainen menettelytapa oli hidas, epäluotettava ja teetti ylimääräistä työtä.

Työ rajattiin määrittelyyn, suunnitelluun ja toteutukseen. Ylläpidosta ja asennuksesta asiakas vastasi itse. Asiakkaan on tarkoitus ottaa sovellus käyttöön mahdollisimman pian.

Tulevaisuudessa varastonhallinta integroidaan toiminnanohjausjärjestelmään. Toiminnanohjausjärjestelmän tarkoitus on nimensä mukaan ohjata yrityksen toimintaa. Sillä pyritään ennen kaikkea tehostamaan yrityksen toimintaa sekä löytämään yrityksen työprosesseista monimutkaisia ja hitaita käytäntöjä, jotka voidaan yksinkertaistaa ja nopeuttaa.

Toiminnanohjausjärjestelmä (ERP-järjestelmä) liittää yhteen yrityksen kannalta monia tärkeitä osa-alueita kuten laskutuksen ja kirjanpidon, tuotannonsuunnittelun, jakelun, varastonhallinnan, asiakas- ja käyttäjähallinnan. Yksittäisinä nämä ominaisuudet eivät ole niin hyödyllisiä yritykselle, mutta kun ne yhdistetään yhdeksi kokonaisuudeksi (Kuva 1), jonka osa-alueet kommunikoivat toistensa kanssa, hyötyy yritys siitä taloudellisesti.


Kuva 1. Toiminnanohjausjärjestelmään kuuluvat osa-alueet. [9.]

## 2 Varastonhallintajärjestelmän määrittely

Tavoitteena oli päästä eroon olemassaolevan prosessin epäluotettavuudesta sekä hitaudesta. Yrityksen kasvaessa talous- ja asiakashallinta kasvavat tärkeämmiksi, eikä nykyinen menetelmä sitoutunut millään tavalla näihin osa-alueisiin. Varastonhallinnan digitalisoiminen on olennaisessa osassa yrityksen kasvun tukemista.

Sovelluksella on tarkoitus hallita varastoon saapuvia ja lähteviä tuotteita sekä ylläpitää yrityksen eri varastoja ja niiden varastopaikkoja. Tuotteita on siis pystyttävä vastaanottamaan sekä toimittamaan. Varastosta löytyvien tuotteiden saldoja on myös pystyttävä inventoimaan. Uusia varastoja sekä varastoihin liittyviä varastopaikkoja pitää pystyä hallinnoimaan. Luotuja varastoja ja niiden varastopaikkoja täytyy pystyä inventoimaan. Kirjanpitolaki edellyttää, että varastoja inventoidaan säännöllisen aikavälein. [1.]

Näiden lisäksi sovellukseen toteutettiin hieman myynti- ja ostoreskontra-ominaisuuksia. Myynti- ja ostotilauksien tekeminen tuli olla mahdollista. Tilauksiin pitää pystyä liittämään tilausrivejä.

Sovellukseen toteutettiin myös tuotehallinta. Aluksi tuotteet koostuvat tuotteen nimestä ja tuotenumerosta, mutta tulevaisuudessa tuotehallintaa laajennetaan.

Ensimmäiseen versioon ei toteutettu käyttäjä- tai asiakashallintaa. Käyttäjähallinnan puuttumisen takia sovellukseen ensimmäiseen versioon ei voi kirjautua sisään. Tästä syystä sovelluksen piti olla käytettävissä ainoastaan lähiverkosta käsin. Kirjanpitoon ja taloushallintaan liittyviä ominaisuuksia ei toteutettu ensimmäiseen versioon.

Sovellus on web-pohjainen eli sitä käytetään selaimella. Ensimmäisessä versiossa mobiililaitteiden selainten rajoituksia ei otettu huomioon. Tietokoneilla sovellus testattiin Mozilla Firefox 19.0 selaimella. Muut selaimet eivät ole tuettujen listalla.

Sovellusta voi ajaa WEBrick-palvelimella. Muut palvelimet eivät ole tuettuja. Palvelimen käyttöjärjestelmän asiakas saa päättää itse.

Mallien ja niiden riippuvuuksien määrittely perustui asiakkaan tarpeiden analysointiin ja muiden varastonhallintasovelluksien tutustumisesta kertyneisiin tietoihin.

Sovelluksen pitää pystyä ylläpitämään yhden yrityksen monen eri varaston varastosaldoja.

Aluksi sovelluksen varasto-osaa tullaan käyttämään ainoastaan tuotteiden toimittamiseen, vastaanottamiseen sekä inventoimiseen. Mallirakenteen tulee mahdollistaa monen tuotteen tulouttaminen samalle varastopaikalle. Rakenne suunniteltiin niin, että varaston varastopaikkahistorian tallentaminen on mahdollista myöhemmässä vaiheessa niin haluttaessa.

# 3 Menetelmät ja työvälineet

## 3.1 MVC-suunnittelumalli

Sovelluksen toteutukseen valitun Rails-sovelluskehyksen suurimpia etuja on, että se toteuttaa yleisimpiä ohjelmointisuunnittelumalleja. Suunnittelumalleilla toteutetun ohjelmakoodin etu verrattuna suunnittelumallittomaan toteutukseen on, että suunnittelumallit yhtenäistivät eri ihmisten tekemää ohjelmakoodia. Jokaisella ohjelmoijalla on ominainen ohjelmointityyli, mutta kun jokainen pyrkii ohjelmoimaan suunnittelumallin mukaisesti, ovat ohjelmakoodit rakenteeltaan hyvin samanlaisia. Tällöin ohjelmoijan on helpompi ymmärtää muiden tekemät ohjelmakoodit.

Kuten monet muutkin web-sovelluskehykset, Rails perustuu MVC- (Model-View-Controller, malli-näkymä-käsittelijä) -suunnittelumalliin. MVC-suunnittelumalli on koko sovelluskehyksen tärkeimpiä osia. Varisinkin web-ohjelmoinnissa ilman sovelluskehystä näkymien (View) ja ohjelmalogiikan (Model ja Controller) sekoittuminen on yksi suurimpia haasteita. Suunnittelumallilla pyritään saamaan rakennetta tähän sekavuuteen. MVC-suunnittelumallin vahvuuksiin kuuluu myös DRY-periaate (Don’t Repeat Yourself). Kun kehittäjä seuraa tätä periaatetta, koodista pitäisi tällöin tulla mahdollisimman vähän itseään toistavaa. Ohjelmakoodi jaetaan eri osiin niiden käyttötarkoituksesta riippuen. Tällöin resurssien toiminnallisuudet kirjoitetaan vain kerran resurssin malliin. Tällöin ne ovat kaikkialla ohjelmakoodin sisällä käytettävissä.

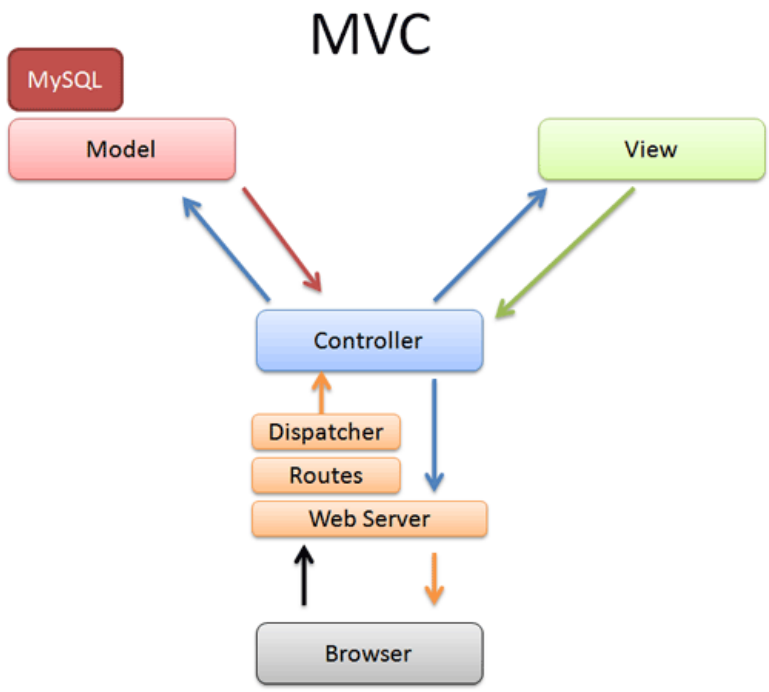
Malli kuvaa sovelluksen dataa ja sääntöjä datan käsittelyä varten. Malleja käytetään pääasiassa tietokantataulujen vuorovaikutussääntöjen hallinnoimiseen. Yleensä jokainen tietokantataulu vastaa yhtä mallia sovelluksessa. Ohjelmalogiikan koko perusta on malleissa. Malliin kirjoitetaan kaikki kyseiseen resurssin liittyvä toiminnallisuus. ActiveRecord on Railsin oletus ORM (Object Relational Mapper). ActiveRecordista periytetään kaikki sovelluksen mallit, jotka halutaan tallentaa tietokantaan.

Näkymät kuvaavat sovelluksen käyttöliittymää. Jokaisella resurssilla on CRUD-toimenpiteiden mukaiset näkymät eri tiedostoissa. Näkymät pyritään pilkkomaan niin pieniin osiin kuin mahdollista, jotta ne olisivat mahdollisimman yleiskäyttöisiä. Näkymiin ei saa tulla resursseihin liittyvää logiikkaa vaan se pitää ohjelmoida malliin. Näkymät

ovat lähes pelkästään HTML-kieltä ja ainoastaan käsittelijän näkymälle tarjoamat resurssit käsitellään ohjelmointikielellä.

Käsittelijät toimivat näkymän ja mallin välillä. Ne ottavat vastaan sisääntulevat kutsut web-selaimelta ja käskevät mallia hakemaan loppukäyttäjän haluaman datan. Tämän jälkeen käsittelijä antaa loppukäyttäjän haluamat resurssit näkymälle, joka käsittelee datan ja esittää sen loppukäyttäjälle. Käsittelijään ei saa ohjelmoida resursseihin liittyvää ohjelmalogiikkaa. Käsittelijän tarkoitus on, että se käskee mallia hakemaan ja käsittelemään halutun datan, minkä jälkeen se annetaan näkymälle näytettäväksi. Käsittelijät tulisi pitää mahdollisimman pieninä ja mallit mahdollisimman suurina.

Kuvassa 2 esitelly englanninkieliset käsitteet vastaavat suomenkielisiä käsitteitä seuraavanlaisetsti. Model tarkoittaa mallia, view näkymää, controller käsittelijää, dispatcher luokkaa, joka osoittaa HTTP-pyynnön reittien osoittamalle käsittelijälle, route reittejä, web server palvelinta ja browser selainta.




Kuva 2. MVC-arkkitehtuurikaavio.

## 3.2 Ruby-ohjelmointikieli

### Yleistä Rubystä

Ruby-ohjelmointikielen on kehittänyt japanilainen Yukihiro "Matz" Matsumoto 1990-luvun puolivälissä. Se on tarkoitettu käyttöliittymäohjelmakoodin tekoon ja siksi se noudattaa käyttöliittymäsuunnittelun periaatteita, joita ovat ytimekkyys, johdonmukaisuus ja joustavuus. Ytimekkyydellä tarkoitetaan, että Rubyllä pystyy antamaan voimakkaita komentoja lyhyesti. Johdonmukaisuudella tarkoitetaan, että Rubyyn on lainattu hyviksi osoittautuneita ominaisuuksia muista ohjelmointikielistä sellaisinaan. Joustavuudella tarkoitetaan, että ohjelmoija voi muokata Rubya vapaasti omien tarpeidensa mukaiseksi. Kielen ydintä ei kuitenkaan voi muuttaa, ja se on pyritty pitämään niin pienenä kuin mahdollista. [2.]

### Rubyn ominaisuudet

Ruby on täysin oliopohjainen tulkattava ohjelmointikieli. Tämä tarkoittaa, että ohjelmakoodi käännetään ihmisen ymmärtämästä ohjelmakoodista tietokoneen ymmärtämäksi koodiksi suorituksen aikana. Tällöin kaikki ulkoisten lähteiden sovelluskehykset ja valmiit kirjastot tarjotaan lähdekoodina. Käytännössä tämä estää suljetun lähdekoodin tekemisen. Suorituksen aikana käännettävän koodin huonona puolena verrattuna valmiiksi konekielelle käännettyyn koodiin on hitaus. Tämän haittapuolen kiertämiseksi on kehitetty YARV (Yet An Other Ruby VM), joka on Ruby-tulkki. [10.]

Kun ohjelma on jäsennelty, luodaan syntaksipuu. Syntaksipuu on hierarkkinen puurakenne kaikista ohjelmassa käytetyistä tokeneista eli tunnisteista, muuttujista ja operaattoreista. Rubyn oletustulkki MRI suorittaa puun läpi, kun Ruby-ohjelma ajetaan. MRI:n lähestymistapa kuluttaa paljon muistia ja on todella tehotonta. YARV lähestyy ongelmaa hieman erillailla. Kun syntaksipuu on luotu, luodaan virtuaalitietokone ohjelmaa varten. Tässä vaiheessa ohjelman koko alkuperäinen rakenne kadotetaan ja ohjelman käskyt esitetään YARV-tulkin haluamalla tavalla (kuva 3). Tämän jälkeen YARV-tulkki suorittaa ohjelmakoodin virtuaalitietokoneessa. [10.]


Kuva 3. YARV-ohjelmakoodin luominen. [11.]

Yleisimmistä tietotyypeistä on Rubyssä oma luokkansa. Tämä tarkoittaa, että mm. kokonaisluvut (integer) ja merkkijonot (string) ovat instansseja niitä vastaavista luokista. Tämä tarkoittaa, että tietotyyppi-olioilla on kutsuttavia metodeja (koodilistaus 1).

```
-119.abs                        # Tulostaa luvun itseisarvon: 199
"tämä on merkkijono".length     # Tulostaa merkkijonon pituuden: 18
```

Koodilistaus 1. Esimerkki kokonaislukujen ja merkkijonojen olioista.

Ruby on vahva dynaamisesti tyypitetty ohjelmointikieli. Ohjelmointikieli on dynaamisesti tyypitetty, kun suurin osa sen tyyppitarkistuksista suoritetaan ajonaikana eikä kääntöaikana. Dynaamisessa tyypityksessä arvoilla on tyyppi, mutta muuttujilla ei. Tämä tarkoittaa, että muuttuja voi viitata minkä tyyppiseen arvoon tahansa. Vahva tyypitys tarkoittaa, että ohjelmointikieli rajoittaa operandien käyttöä erityyppisten arvojen välillä. Jos kokonaisluku on 5 ja merkkijono 14 ja ne yritetään yhdistää toisiinsa + -operandin avulla, vahva tyypitys ei anna tehdä tätä.

Ruby voi ajonaikana ohjelmallisesti muokata omaa rakennettaan, kuten luokkia ja metodi kutsuja. Tämän tyyppisen metaohjelmoinnin (metaprogramming) avulla voimme lyhyesti ja ytimekkäästi laajentaa Rubyn olemassa olevaa koodia. String-luokkaa voidaan esimerkiksi laajentaa ja luoda sille uusia metodeja (koodilistaus 2).

```
COLORS = { black:   "000",
           red:     "f00",
           green:   "0f0",
           yellow:  "ff0",
           blue:    "00f",
           magenta: "f0f",
           cyan:    "0ff",
           white:   "fff" }

class String
  COLORS.each do |color,code|
    define_method "in_#{color}" do
      "<span style=\"color: ##{code}\">#{self}</span>"
    end
  end
end
```

Koodilistaus 2. Metaohjelmointiesimerkki.

Koodilistauksen 2 ohjelmakoodi luo kahdeksan uutta metodia Rubyn String-luokalle: black, red, green, yellow, blue, magenta, cyan ja white. Näiden uusien metodien avulla HTML-sivulle tulevan tekstin väriä voi muuttaa kutsumalla halutulle merkkijonolle värinvaihtometodia `"Hello, World!".in_blue`.

Rubyssä saanti- ja asetusmetodit on ohjelmoitu metaohjelmointi tyylillä. Ohjelmoijalla on käytössään käskyt `attr_accessor`, `attr_reader` ja `attr_writer`. `Attr_accessor` tarkoittaa, että jäsenmuuttujalle luodaan saanti- sekä aseteusmetodi, `attr_reader` tarkoittaa, että jäsen muuttujalle luodaan pelkästään saantimetodi ja `attr_writer` tarkoittaa, että muuttujalle luodaan pelkästään asetusmetodi. Koodilistauksen 3 luokassa on esitelty kyseisten metaohjelmointikäskyjen käyttöä.

```
class User
        attr_accessor :name
        attr_reader :created_at
        attr_writer :age
end
```

Koodilistaus 3. Saanti- ja asetusmetodiesimerkki.

## 3.3 Rails

Ruby on Rails (Rails) on web-ohjelmointisovelluskehys Ruby-kielelle. Sen on kehittänyt David Heinemeier Hansson Basecamp-ohjelman pohjalta. [4.] Hansson julkaisi ensimmäisen version lähdekoodista heinäkuussa 2004.

### Metaohjelmointi

Rubyn metakomentojen lisäksi Railsin ActiveRecord OR -mapping luokka (ORM) sisältää komennot `attr_accessible` ja `attr_protected`. Kun jäsenmuuttujan eteen laitetaan jompikumpi näistä komennoista, tarkoittaa se, että ne tullaan tallentamaan tietokantaan. `Attr_accessible`-komento eroaa `attr_protected`-komennosta siten, että se vapauttaa jäsenmuuttujan niin sanotulle MassAssingment-toiminnolle. Tällöin muuttujan arvo voidaan muuttaa suoraan web-formin requestista (koodilistaus 4).

```
class User < ActiveRecord::Base
        attr_accessible :name, :created_at, :age
        attr_protected :role
end

@user = User.new(params[:user])
```

Koodilistaus 4. Rails metaohjelmointi –esimerkki.

Viimeisellä rivillä luodaan uusi User-olio suoraan web-formin pyynnöstä. Jos User-olion jäsenmuuttuja role olisi määritelty `attr_accessible :role`, tarkoittaisi tämä, että sen arvoa pystyttäisiin muuttamaan MassAssingment-toiminnolla. Tämä on sovelluksessa paha tietoturvariski eli role-jäsenmuuttujaa ei määritellä `attr_accessible`-komennolla. Hyökkääjä voisi injektoida Javascriptillä User-olion luonti-web-lomakkeeseen uuden syötekentän nimeltään role. Tällöin hyökkääjä voisi määrätä luomansa käyttäjän roolin. Tästä syystä role-jäsenmuuttuja pitää olla määritelty kommennolla attr_protected, jolloin sitä ei aseteta komennossa `@user = User.new(params[:user]`).

### 3.4 RubyGem

RubyGem on paketinhallintatyökalu Ruby-kielelle. Se tarjoaa standardityylin Ruby-ohjelmien ja -kirjastojen levittämiseen.

RubyGemien avulla on hyvin helppo muuttaa tai laajentaa olemassa olevaa Ruby-sovellusta. RubyGemien tunteminen on todella hyödyllistä, koska niihin sisältyy paljon yleisiä toiminnallisuuksia. Jos kehittäjällä on jokin ongelma, kannattaa aina ensin tarkastaa, onko joku jo ratkaissut kyseisen ongelman ja tehnyt siitä RubyGemin.

Koska Ruby on suorituksen aikana käännettävää kieltä, RubyGemit on tarjottava lähdekoodina. Tästä syystä Gemit yleensä löytyvät Github-palvelusta, jolloin ne ovat kaikkien vapaasti muokattavissa.

### 3.5 Ruby Bundler

Ruby Bundlerin tarkoitus on ylläpitää yhtenäistä ympäristöä Ruby-ohjelmille. Se seuraa ohjelmakoodia ja ohjelmakoodiin tarvittavien RubyGemejä, jotta suoritettavalla ohjelmalla on aina sen vaatimat Gemit ja niistä oikeat versiot.

Jokaisella Rails-sovelluksella on oma asennuksensa. Asennustiedostot sijaitsevat projektikansiossa siinä polussa johon sovellus on asennettu `rails new` –komennolla. Tarvittavat tiedostot ja niiden versiot määräytyvät Gemfilen perusteella. Sovelluskohtaisten asennus tiedostojen takia eri sovelluksilla voi olla eri versiot Rails-kirjastoista. Bundlerin `install`-komennolla tarvittavat kirjastot voidaan asentaa projektikansioon (koodilistaus 5).

```
$ bundle install
Using rake (10.0.2)
Using multi_json (1.3.7)
Using activesupport (3.2.9)
Using builder (3.0.4)
Using activemodel (3.2.9)
Using erubis (2.7.0)
Using journey (1.0.4)
Using rack (1.4.1)
Using rack-cache (1.2)
Using rack-test (0.6.2)
Using hike (1.2.1)
Using tilt (1.3.3)
Using sprockets (2.2.1)
Using actionpack (3.2.9)
Using mime-types (1.19)
Using polyglot (0.3.3)
Using treetop (1.4.12)
Using mail (2.4.4)
Using actionmailer (3.2.9)
Using arel (3.0.2)
Using tzinfo (0.3.35)
Using activerecord (3.2.9)
Using activeresource (3.2.9)
Using afm (0.2.0)
Using coffee-script-source (1.4.0)
Using execjs (1.4.0)
Using coffee-script (2.2.0)
Using rack-ssl (1.3.2)
Intalling json (1.7.5)
Your bundle is complete! It was installed into ./vendor/bundle
```

Koodilistaus 5. Bundler-komennon ajaminen.

Bundler asentaa vain vendor/bundle-kansiosta puuttuvat RubyGemit. Varhaisempia asennuksia ei uudelleenasenneta.

### 3.6 Rake

Rake on Ruby-ohjelmointikielelle ohjelman tehtävänhallintatyökalu. Sen vastineita UNIX-järjestelmässä on make ja Javassa Apache Ant. Tehtävänhallintatyökalun tarkoituksena on automatisoida erilaisia tehtäviä, joita ohjelmoijat kohtaavat päivittäin. Nämä tehtävät voivat olla muun muassa testien ajamista, ohjelmakoodin kääntämistä tuotantoympäristöön tai dokumentaation luomista ohjelmakoodin kommenttien perusteella. Rakea käytetään siis erilaisten tiedostojen luomiseen automaattisesti. Rake tarvitsee toimiakseen sääntötiedoston, jossa kuvataan halutut tiedostot sekä niiden riippuvuudet muihin tiedostoihin. Rakelle pitää myös kertoa mitä tiedostojen kuuluu tehdä. Oletuksena Rake etsii näitä sääntöjä Rakefile-nimisestä tiedostosta. [5.]

Railsissä Rakea käytetään muun muassa olioiden synkroinoimisessa tietokannan kanssa. Railsin avulla Rakelle luodaan migraatiotiedostoja, joiden avulla tietokantataulut päivitetään.

### 3.7 Javascript-kirjastot

Sovelluksen kehittämiseen käytettiin suosittua jQuery-kirjastoa. Kirjaston tarkoituksena on yksinkertaistaa käyttäjäpuolen (client-side) ohjelmointia. jQuery on julkaistu tammikuussa 2006 ja sen on kehittänyt John Resig. Tällä hetkellä jQueryä kehittää Dave Methvin johtama ryhmä. jQuery on maailman suosituin Javascript-kirjasto.

jQueryn synktaksi on suunniteltu helpottamaan HTML-dokumentin sisällä navigoimista, DOM-elementtien valintaa, animaatioiden luomista, tapahtumien hallintaa (event handle) sekä AJAX-kutsujen luomista. [6.]

jQuery tarjoaa ohjelmoijalle mahdollisuuden liitännäisten (plug-in) tekemiseen jQuery-kirjaston päälle. Tämä mahdollistaa matalan tason abstraktioiden toteutuksen käyttöliittymän vuorovaikutuksen ja animaatoiden kanssa. Ohjelmoija voi siis kehittää liitännäisiä, jotka voidaan liittää mille tahansa web-sivulle. Hyvä esimerkki tällaisesta liitännäisestä on jQueryn Autocomplete-liitännäinen, jonka voi liittää mihin tahansa HTML-syötekenttään. Autocomplete-kirjaston ideana on, että se hakee tietokannasta tai muusta ulkoisesta lähteestä käyttäjän syötekenttään syöttämän tekstin perusteella tietoa ja ehdottaa haettujen tietojen perusteella käyttäjälle vaihtoehtoja.

Dynaamisen web-sisällön luominen on myös nykyaikaisessa web-ohjelmoinnissa hyvin tärkeää. Tällä tarkoitetaan, että HTML-dokumentin sisältö muuttuu käyttäjän valintojen mukaan. Ilman Javascriptiä käyttäjän valinta pitäisi lähettää HTTP-pyyntönä palvelimelle, jossa se käsiteltäisiin ja uusi sisältö renderöitäisiin selaimelle. Tällainen menettelytapa on hyvin kankea ja hidas. Javascriptillä uusi sisältö voidaan suoraan luoda käyttäjän päässä HTML-dokumenttiin, eikä sivun uudelleenlautauksia siis tarvita.

### 3.8 HTML ja CSS

Sovelluksen käyttöliittymän toteutukseen käytettiin HTML:ää ja CSS:ää. HTML eli HyperText Markup Language on kuvauskieli, jolla kuvataan sisällön rakennetta ja esitystapaa. Selaimet käsittelevät HTML-elementtejä ja luovat niiden perusteella näkymän web-sivusta. HTML muun muassa tukee tekstiä, kuvia, videoita ja flash-sovelluksia. HTML-standardi on World Wide Web Consortiumin (W3C) hallinnoima. [12.]

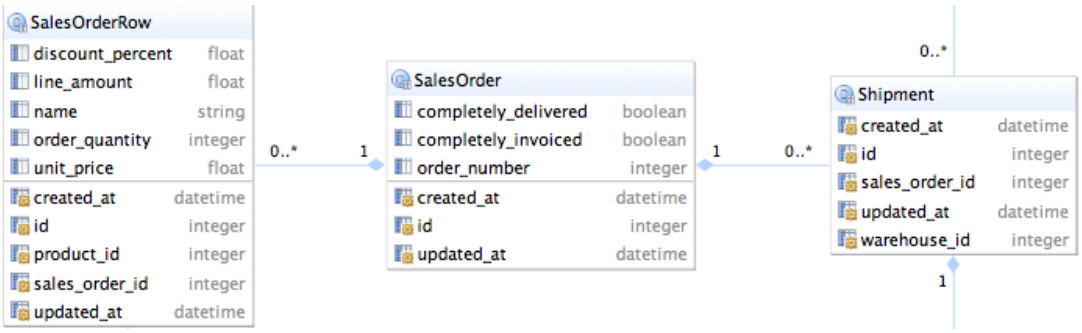
CSS eli Cascading Style Sheets on tyylimerkintäkieli. CSS:n avulla kirjoitetaan tyyliohjeita, jotka kuvaavat miltä HTML-sivun pitäisi näyttää. CSS tarjoaa mahdollisuuden muokata web-sivujen ulkonäköä muokkaamatta sivun sisältöä tai rakennetta. Kuten HTML, myös CSS on W3C:n hallinnoima. [30.]

### 3.9 MariaDB

MariaDB on MySQL-yhteisön luoma haara MySQL-tietokantajärjestelmästä. MariaDB on kehitetty, koska MySQL-tietokannan lisenssivapauden tulevaisuus ei ollut enää taattu. MariaDB pohjautuu MySQL:ään, jonka ensimmäinen versio on julkaistu 23. toukokuuta 1995. MariaDB on julkaistu 22. tammikuuta 2009. [7.]

MariaDB on relaatiotietokanta eli sen datan tallennus perustuu relaatiomalliin, jonka on ensimmäisen kerran esitellyt Edgar F. Codd vuonna 1970 [8]. Relaatiomallissa data esitellään tauluina, jotka yhdistetään toisiinsa (Kuva 4).

Relaatiomallin ideana on tarjota selkeä tyyli datan tallennusta ja hakua varten. Relaatiotietokantaan kuvataan tallennettu data tauluihin sellaisena kuin se halutaan tallentaa. Tauluun tallennetaan tietyn käsitteen arvoja eri sarakkeisiin ja mahdollisia relaatioita omiin sarakkeihinsa. Taulujen välisten suhteiden avulla voidaan yhdistellä eri tauluja toisiinsa ja näin hakea yhteenkuuluvia kokonaisuuksi SQL-hakukomennolla (query). SQL-komennoilla voidaan myös lisätä tietoa tai päivittää taulussa olevaa dataa.




Kuva 4. Esimerkki relaatiotietokannan tauluista.

# 4 Järjestelmän suunnitelu

## 4.1 Varastonhallinnan käsitteet

Sovelluksen suunnittelu aloitettiin määrittelyn pohjalta syntyneiden käsitteiden ja asiakkaan tarpeiden analysoimisella. Asiakas tarvitsi järjestelmän, jolla hallinnoidaan osto- ja myyntitilauksia, saapuvia ja lähteviä toimituksia, varastoja ja varastopaikkoja sekä varastoissa olevia tuotteita.

Ostotilaus tarkoittaa varastoon tilattavaa tuotelistaa halutulta tavarantoimittajalta. Ostotilaukseen liitetään tavarantoimittaja ja toimittajalta tilattavat tuotteet. Tuotteet liitetään ostotilaukseen ostotilausriveinä, jotka pitävät sisällään tilatun kappalemäärän.

Myyntitilauksella tarkoitetaan varastosta lähtevien tuotteiden listaa. Myyntitilaukseen liitetään asiakas, joka on tilannut järjestelmästä löytyviä tuotteita. Tilatut tuotteet ja kappalemäärät liitetään myyntitilaukseen myyntitilausriveinä.

Saapuminen on tiettyyn varastoon saapunut toimitus. Saapumiseen liitetään saapuneet tuotteet ja kappalemäärät. Saapumiset lisäävät varastojen tuotteiden varastosaldoja. Tällöin saapumisen perusteella luodaan varastotapahtumia, jotka lisäävät varastosaldoa saapumisrivikohtaisesti. Saapuminen siis kuvaa tuotteiden fyysistä saapumista varastoon.

Lähtevät toimitukset luodaan myyntitilauksien perusteella. Lähteviin toimituksiin liitetään varastosta kerätyt tuotteet sekä kerätyt kappalemäärät. Lähtevät toimitukset vähentävät tuotteiden varastosaldoa. Toimitusten perustella luodaan varastotapahtumia, jotka vähentävät varastosaldoa toimitusrivikohtaisesti. Toimituksilla kuvataan tuotteiden fyysistä poistumista varastosta.

Saapuvat ja lähtevät toimitukset liitetään aina tiettyyn varastoon. Tästä syystä järjestelmästä pitää löytyä varasto-käsite, joka kuvaa yrityksen tiettyä varastoa. Varastoihin liittyy myös varastopaikka, joka kuvaa tiettyä varastopaikkaa varastossa.

Varastojen varastopaikkojen tuotteiden kappalemäärä vaihtuu saapuvien ja lähtevien toimitusten perusteella. Kappalemääriä muutetaan todellisuutta vastaavaksi keräyksessä tai vastaanotossa tapahtuneiden virheiden takia. Varastoinventointi-käsite kuvaa tätä prosessia. Varastoinventoinnissa inventoidaan varaston varastopaikkoja.

Varastoinventointiin liitetään inventointirivejä, jotka muuttavat varastopaikan saldoa varastopaikalta laskettujen tuotteiden kappalemäärän perustella.

Näiden käsitteiden perusteella voitiin suunnitella käyttöliittymä ja oliomallit.

## 4.2 Käyttöliittymä

Varastonhallintasovelluksen ensimmäisen version käyttöliittymä suunniteltiin mahdollisimman yksinkertaiseksi. Käyttöliittymän ulkoasuun ei panostettu, koska toiminnanohjausjärjestelmän luonteen takia se ei ollut merkittävää.

Myynti- ja ostotilausten tekemisen on oltava mahdollisimman helppoa. Tilauksiin liitettävien tilausrivien syöttäminen toteutettiin Autocomplete jQuery -pluginin avulla.

Ensimmäisessä versiossa saapuvien ja lähtevien toimitusten näkymät luodaan suoraan tilausten pohjalta. Näkymistä syötetään tilausten perusteella saapuneet tai lähtevät kappalemäärät.

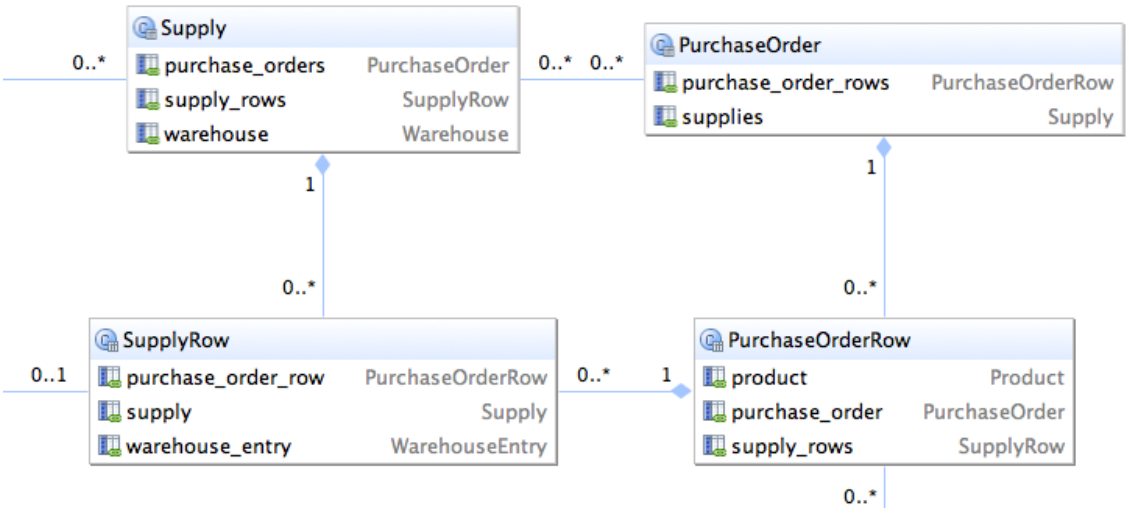
Varastoinventointinäkymästä inventoidaan varastopaikkoja varastopaikkakohtaisesti. Järjestelmä esitäyttää varastopaikalta löytyvän kappalemäärän ja käyttäjän pitää syöttää todellisuutta vastaava kappalemäärä, minkä avulla tuotteen kyseisen varastopaikan varastosaldot korjataan.

Koska sovellus on web-pohjainen, käyttöliittymä toimii myös mobiililaitteilla. Päätelaitteiden erilaisuus aiheuttaa käyttöliittymäsuunnitteluun haasteita. Ensimmäisen version käyttöliittymää ei suunniteltu mobiililaitteille eikä eri resoluutioille skaalautuvaksi. Käytössä on Rails-sovelluskehyksen Gem-lisäosista löytyvä automaattinen sommittelu.

## 4.3 Oliomallit

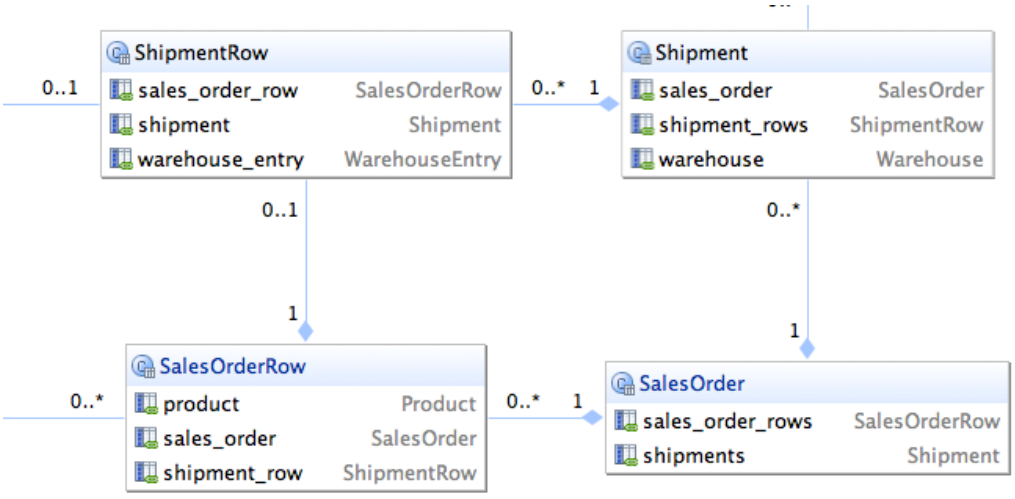
Myyntitilaus (SalesOrder) (kuva 6) ja ostotilaus (PurchaseOrder) (kuva 5) ylläpitää tilaustietoja tilausrivikohtaisesti (SalesOrderRow ja PurchaseOrderRow). Tilausriville tallennetaan tieto tilatusta tuotteesta, tilattu määrä sekä myynti- tai ostohinta. Myyntitilauksella tallennetaan myös mahdollinen alennusprosentti. Myynti- ja ostotilaukset ovat osa myynti- ja ostoreskontraa eivätkä suoranaisesti liity varastonhallintaan.

Kun varastoon saapuu fyysisesti tuotteita, ne liittyvät aina johonkin ostotilaukseen. Tällöin luodaan uusi saapuminen (Supply) sekä ostotilausrivien perusteella saapumisrivit (SupplyRow) (kuva 5). Saapumisriveille kirjataan saapuneiden tuotteiden määrä, joka ei välttämättä vastaa tilattujen tuotteiden määrää.




Kuva 5. Ostotilaus-, ostotilausrivi-, saapumis- ja saapumisrivimallin riippuvuudet toisistaan.

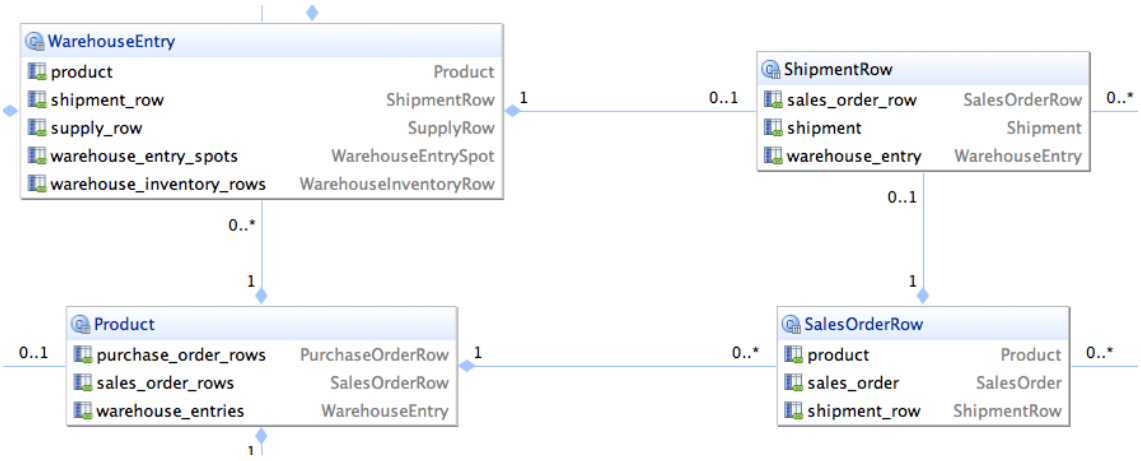
Varastoon tulouttamisen jälkeen tuotteet löytyvät varastosaldoilta. Tällöin ne voidaan toimittaa myyntitilauksien perusteella asiakkaille. Kun myyntitilauksen tuotteet halutaan fyysisesti toimittaa asiakkaalle, tehdään tilatuista tuotteista toimitusrivit (ShipmentRow), jotka liitetään toimitukseen (Shipment) (kuva 6). Toimituksia ei voida tehdä, jos varaston saldoilla ei ole haluttua tuotetta.




Kuva 6. Myyntitilaus-, myyntitilausrivi-, toimitus ja toimitusrivimallin riippuvuudet toisistaan.

Varastotapahtumarivi (WarehouseEntry) luodaan aina, kun varastoon saapuu tai varastosta lähtee tuotteita. Saapumisrivin ja toimitusrivin riipuvuus varastotapahtumaa on yhden suhde yhteen. Tämä tarkoittaa, että tapahtuma sisältää täsmälleen saman määrän tuotteita kun saapumis- ja toimitusrivit. Varastotapahtuma on keskeisessä osassa varaston toimintaa. Ilman sitä tuotteita olisi mahdoton inventoida.

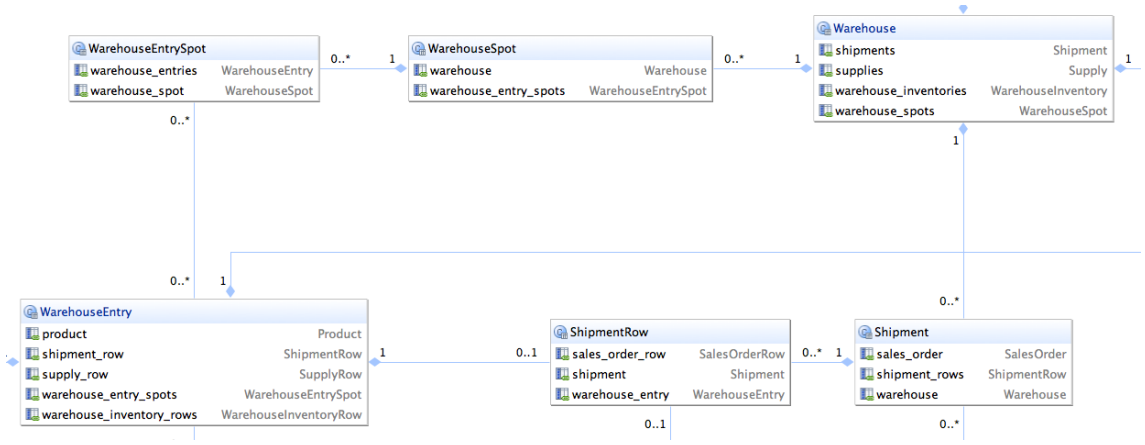
Tuote (Product) liittyy ostotilausriveihin ja myyntitilausriveihin. Tilausrivien riippuvuus tuotteista on yhden suhde moneen. Tämä tarkoittaa, että tilausrivillä on yksi tuote ja tuotteella voi olla monta tilausriviä. Kun tilauksien perusteella vähennetään varastosaldoja, liitetään varastotapahtumaan myös tuote. Varastotapahtuman ja tuotteen suhde on yhden suhde moneen (kuva 7). Tuotteita ei olisi välttämätöntä liittää varastotapahtumaan, koska varastotapahtuman tuote saataisiin selville varastotapahtuman toimitusrivin tilausrivin tuotteesta (kuva 7). Tässä tilanteessa monta eri taulua liitetään toisiinsa, jolloin tietokantakysely on hidas. Tästä syystä varastotapahtumalle liitetään tuote varastosaldojen kirjaamisvaiheessa.




Kuva 7. Tuotemallin riippuvuus varastotapahtumista ja myyntitilausriveistä.

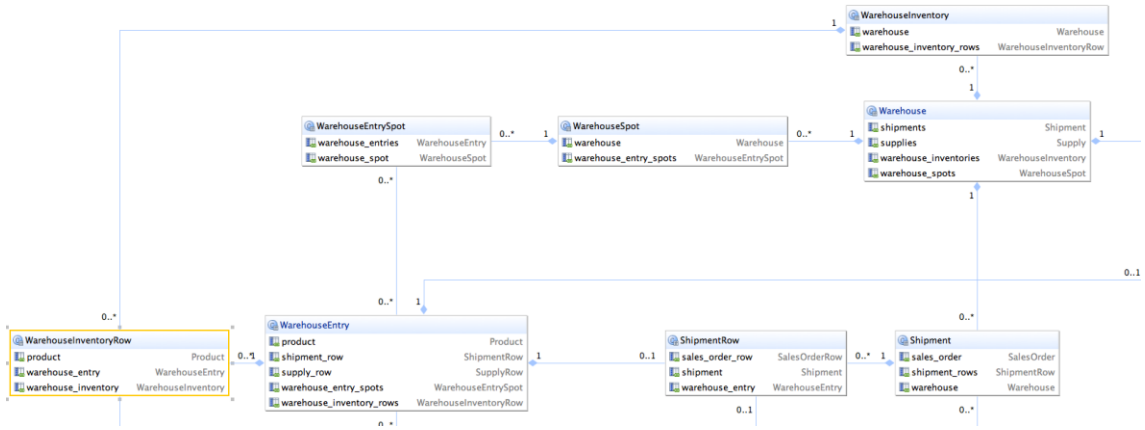
Varastotapahtumapaikka (WarehouseEntrySpot) yhdistää varastotapahtuman varaston varastopaikalle (WarehouseSpot) (kuva 8). Varastotapahtuman ja varastopaikan välillä on oltava ne yhdistävä malli, joka pitää yllä varastopaikalle laitettujen tuotteiden lukumäärää. Varastotapahtuma ei voi olla suoraan linkitetty varastopaikkaan, koska tällöin saapumisrivi voitaisiin yhdistää vain yhteen varastopaikkaan. Jos tilatut tuotteet ovat esimerkiksi monella varastolavalla, ei niitä ole fyysisesti mahdollista laittaa samalle paikalle.

Varastopaikka (WarehouseSpot) sisältää varaston kaikki varastopaikat. Varastopaikkaan on yhdistetty varasto (Warehouse) (kuva 8). Yksi varastopaikka voi kuulua vain yhteen varastoon, ja yhdellä varastolla voi olla monta varastopaikkaa. Niiden välinen riippuvuus on siis varaston näkökulmasta yhden suhde moneen.




Kuva 8. Varasto-, varastopaikka-, varastotapahtuma-, toimitusrivi- ja toimitusmallien riippuvuudet toisistaan.

Jotta varastosaldoja pystytään korjaamaan, täytyy olla työkalu sitä varten. Varastoinventointi (WarehouseInventory) on tätä varten. Se sisältää varastoinventointirivejä (WarehouseInventoryRow) (kuva 9), joihin tallennetaan varastosaldopoikkeamia.




Kuva 9. Varastomallien ja toimitusmallien riippuvuudet toisistaan.

Edellä mainittujen käsitteiden pohjalta suunniteltiin malliriippuvuuskaavio, joka kuvaa mallitasolla järjestelmän toimintaa. Malliriippuvuuskaavio vastaa hyvin pitkälti relaatiokaaviota. Oliopohjaisessa suunnittelussa ei kuitenkaan haluta keskittyä tietokannan suunnitteluun vaan olioiden välisten riippuvuuksien määrittelyyn. Oliopohjaisessasuunnittelumallissa ei olla kiinnostuneita niinkään tietokantarakenteesta, koska tietokanta on vain datan säilytystä varten eikä se saisi vaikuttaa olioiden riippuvuuksien määrittelyyn.

Kertyneiden tietojen pohjalta voidaan piirtää malliriippuvuuskaavion (liite 1).

## 4.4 Tietomalli

Railsin MVC-arkkitehtuurisessa suunnittelussa tietokantaa ei tarvitse suunnitella erikseen vaan se luodaan oliomallien pohjalta. Oletuksena jokainen oliomalli vastaa yhtä tietokantataulua. Oliomallin instanssimuuttujat vastaavat tietokantataulun sarakkeita.

Oliomallien pohjalta luotavat tietokantataulut luodaan oletuksena oliomallin nimeä vastaavaa monikkoa. Product-olion tietokantataulu on siis products. Oliomallin instanssimuuttujat nimetään tietokantasarakkeisiin samannimisiksi.

MVC-arkkitehtuurin mukainen suunnitelu nopeuttaa sovelluksien toteutusta, koska sovelluksien toteutuksessa ei tarvitse ottaa huomioon tietokantaa huomioon lainkaan. Tietokannat luodaan Convention over configuration -periaatteen mukaisesti, mikä tarkoittaa, että ohjelmoijan ei tarvitse käyttää resursseja lähdekoodin kannalta merkityksettömämpien osa-alueidein toteutukseen. [14.]

Tietokantaa käytetään vain tiedon tallentamiseen ja sen hakemiseen. Se ei saisi siis vaikuttaa itse sovelluksen kehitykseen. Tietokantateknologia voi vaihtua taustalla, mutta sovelluksen lähdekoodin ei tarvitse muuttua.

# 5 Sovelluksen toteutus

Suunnittelun jälkeen aloitettiin ohjelman toteutus suunnitelman perusteella. Sovelluksen pohja luotiin Railsin komentorivityökalulla käskyllä: `new`.

```
$ rails new wamas
```

Tämä komento luo uuden kansion nimeltään wamas ja sen alle Gemfilen määrittelemien RubyGemien perusteella sovelluskehyksen tarvitsemat kansiot ja tiedostot (liite 3). Komento ajaa myös `bundle install` -komennon, joka asentaa haetut Gemit. Tämän jälkeen luodaan Raken avulla tietokannan, jota sovellus käyttää.

```
$ rake db:create
```

Kun tietokanta on luotu voimme käynnistää WEBrick palvelimen (koodilistaus 6).

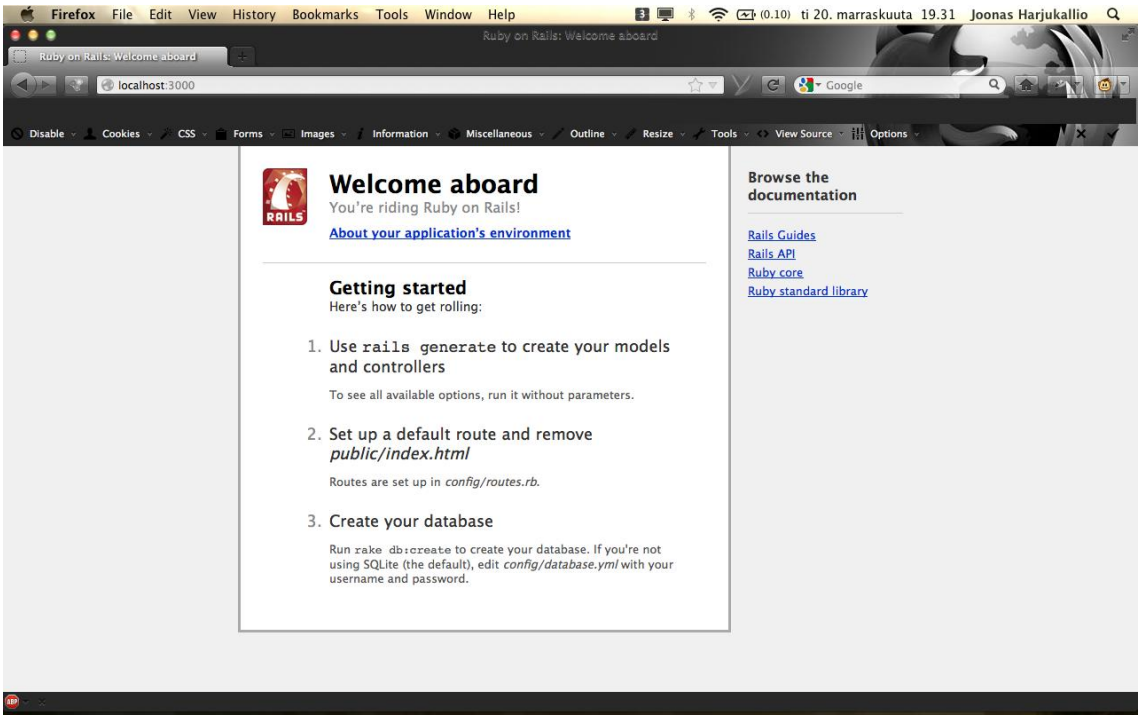
```
$ rails server
=> Booting WEBrick
=> Rails 3.2.9 application starting in development on
http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
[2012-11-20 19:26:09] INFO  WEBrick 1.3.1
[2012-11-20 19:26:09] INFO  ruby 1.9.3 (2012-04-20) [x86_64-
darwin11.4.2]
[2012-11-20 19:26:09] INFO  WEBrick::HTTPServer#start: pid=4227
port=3000
```

Koodilistaus 6. Railsin vakiopalvelimen käynnistys.

Tämän jälkeen sovellukseen pitää luoda ensimmäinen käsittelijä, joka ottaa vastaan juuripolkuun tulevat pyynnöt.

```
$ rails generate controller home index
```

Voidaan todeta, että palvelin on pystytetty oikein ja että se on toiminta kunnossa menemällä selaimella juuripolkuun (kuva 10).




Kuva 10. Ruutukaappaus Rails-palvelimen asennuksen jälkeisestä tilanteesta.

Ohjelman toteutus aloitettiin luomalla suunnitelman pohjalta luokat ja niiden väliset riippuvuudet (liite 1). Railsissä uudet luokat luodaan aina Railsin komentorivityökalulla. Ohjelmoijalla on käytössään seuraavanlaiset komennot:

- generaten avulla voidaan luoda muun muassa runkoja (scaffold), käsittelijöitä ja malleja.
- serverin avulla voidaan käynnistää Rails-palvelin.
- new:n avulla voidaan luoda uusi Rails-sovellus.
- destroyn avulla voidaan poistaa generate-komennolla luotuja runkoja, käsittelijöitä tai malleja.

## 5.1 Uusi runko

Aloitetaan luomalla tuotteille uusi runko (scaffold). Tämä tarkoittaa, että Rails luo halutun käsittelijän, mallin, näkymät, migraatio-tiedostot, reitit ja testit yhdellä komentorivikomennolla (liite 4).

```
$ rails generate scaffold Product name:string
date_created:datetime date_modified:datetime deleted:boolean de-
scription:text
```

Rails luo komennon pohjalta migraatiotiedoston, joka luo tietokantaan uuden taulun ja siihen halutu sarakkeet (koodilistaus 7).

```
create_products.rb:

class CreateProducts < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string :name
      t.datetime :date_created
      t.datetime :date_modified
      t.boolean :deleted
      t.text :description

      t.timestamps
    end
  end
end
```

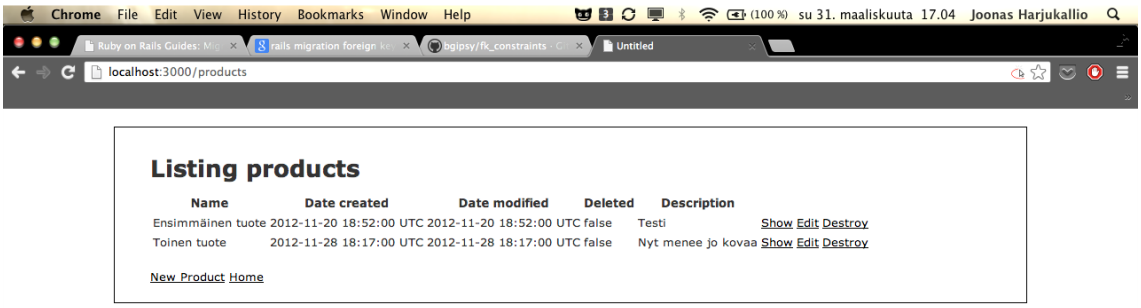
Koodilistaus 7. Rails-migraatiotiedosto.

CreateProducts-luokka luo sovelluksen tietokantaan uuden taulun ja siihen halutut kentät. Migraatiotiedostot pitää päivittää tietokantaan Raken avulla, ajamalla komento rake db:migrate (koodilistaus 8).

```
$ rake db:migrate
==  CreateProducts: migrating
================================================
-- create_table(:products)
   -> 0.1150s
==  CreateProducts: migrated (0.1152s)
=======================================
```

Koodilistaus 8. Migraatioiden ajaminen tietokantaan.

Tuote-luokka on luotu sekä kaikki tarvittavat metodit. Tuoteet-resurssi on käyttövalmis, menemällä osoitteeseen http://localhost:3000/products (kuva 11). Kuvassa näkyy myös kaksi käsin lisättyä tuotetta.




Kuva 11. Products-näkymä.

Railsin scaffold-komento on siis hyvin tehokas, kun halutaan luoda resurssille CRUD-toiminnallisuus ja siihen liittyvät näkymät sekä testit automaattisesti.

## 5.2 Uusi malli ja kontrolleri

Hyvin usein ohjelmoija ei kuitenkaan halua käyttää scaffold-komentoa sen liiallisen automaattisuuden takia. Siitä syystä ohjelmoija voi myös luoda komentorivityökalun avulla käsittelijöitä ja malleja. Luodaan myyntitilaus ja sille myyntitilausrivi.

```
$ rails generate model SalesOrder order_number:integer complete-
ly_delivered:boolean completely_invoiced:Boolean
```

```
$ rails generate controller SalesOrder
```

```
$ rails generate model SalesOrderRow name:string
row_number:integer order_quantity:integer unit_price:float dis-
count_percent:float discount_amount:float line_amount:float
unit_cost:float sales_order:references

$ rake db:migrate
```

Edelläolevien komentojen avulla luotiin myyntitilauksen ja myyntitilausrivin, joka liittyy myyntitilaukseen. Myyntitilausrivien suhde myyntitilauksiin määritellään komennolla `sales_order: references`.

## 5.3 Uusi migraatio

Kun sovellus kasvaa ja siihen tulee uusia malleja, joitain aikaisempia malleja täytyy muokata, jos niiden välille halutaan suhde. Tällöin täytyy tehdä uusi migraatio, jossa määritellään halutut suhde. Migraatio luodaan komennolla `rails generate migration`.

```
$ rails generate migration
add_product_id_to_sales_order_row product_id:integer
```

Kun migraatio on nimetty tiettyjen sääntöjen mukaan, osaa Rails päätellä, mihin tauluun halutaan lisätä, poistaa tai muuttaa ja mitä halutaan lisätä. Edelläoleva käsky siis lisää (add) tuote-id:n (product_id) myyntitilausrivitauluun (to_sales_order_row), joka on tyyppiä kokonaisluku (product_id:integer). (koodilistaus 9)

```
class AddProductIdToSalesOrderRow < ActiveRecord::Migration
  def change
    add_column :sales_order_rows, :product_id, :integer
  end
end
```

Koodilistaus 9. Luodun migraatiotiedoston sisältö.

Kun ohjelmoija osaa käyttää Railsiä ja ohjelmoi niin kuin Rails on suunniteltu, ohjelmoija säästää hyvin paljon aikaa uusien sovelluksien toteutuksessa.

## 5.4 Javascript-liitännäiset

Eri resursseihin liittyvät Javascript-liitännäiset haluttiin rakentaa niin, että ne ovat myös käytettävissä omien resurssiensa ulkopuolella. Tämä tarkoittaa, että liitännäiset

suunniteltiin niin, että ne ovat yleiskäyttöisiä eivätkä ratkaise pelkästään yksittäistä ongelmaa.

Tuotteen varastosaldon hakeminen dynaamiseen sisältöön tarvitsee Javascript-rajapinnan, joka hakee kyseisen tuotteen saldon palvelimelta tuotteet-resurssilta (koodilistaus 10).

```
(function ($) {
    var ProductPlugin = function (element) {
        var element = $(element);
        var obj = this;

        this.fetchBalance = function (warehouse_id, product_id)
{
            return $.ajax({
                async:false,
                dataType:'json',
                type:'GET',
                url:'/products/balance/' + warehouse_id + '/' +
product_id
            }).done(function (data) {
                    if (console && console.log) {
                        console.log('Balance was fetched');
                        console.log(data);
                    }
                });
        };
    };

    $.fn.productPlugin = function () {
        return this.each(function () {
            var element = $(this);

            if (element.data('productPlugin')) {
                return;
            }

            var productPlugin = new ProductPlugin(this);

            element.data('productPlugin', productPlugin);
        });
    };

})(jQuery);
```

Koodilistaus 10. jQuery-liitännäinen.

Ohjelmassa laajennetaan jQuery-kirjastoa ja jQueryyn lisätään uusi funktio productPlugin, joka luo uudet ProductPlugin olion selektorin määräämiin HTML-elementteihin, jos niitä ei ole aikaisemmin luotu. ProductPlugin-olio sisältää yhden

julkisen funktion nimeltään fetchBalance, joka hakee Rails-palvelimelta tuotteen varastosaldon, varaston ja tuote-id:n perusteella products-resurssilta. Koska Rails toteuttaa REST-mallia, tuotteen saldo haetaan polusta /products/balance/:warehouse_id/:product_id. REST-mallissa products tarkoittaa käsittelijää ja ”/”-merkin jälkeen tuleva osa käsittelijän metodia, joka on balance. Metodin ”/”-merkin jälkeen tulee sille annettavat parametrit. Tässä tapauksessa ne ovat varasto-id ja tuote-id. Kun käyttäjäpuolen (client-side) ohjelmat suunnitellaan näin, niitä voidaan kutsua helposti (koodilistaus 11).

```
$('#product_balance').productPlugin();
var product_plugin =
$('#product_balance').data('productPlugin');
var product_request_obj = prod-
uct_plugin.fetchBalance(warehouse_id, product_id);
product_request_obj.done(function(fetched_balance){
  $('#product_balance').val(fetched_balance.balance);
});
```

Koodilistaus 11. jQuery-liitännäisen kutsuminen.

Palvelinpuolella tuotteen varastosaldonhakumetodi hakee tuotteiden varastotapahtumien, varastopaikkojen ja varaston avulla tuotteen varastosaldon. Funktiolle annetaan HTTP GET -kutsun parametreina tuotteen id sekä varaston id. Product-malli käyttää näitä tietoja tietokantakyselyssä ja palauttaa tuotteen id:n nimen ja varastosaldon. Tuote-malli muutetaan JSON-olioksi ja palautetaan Javascriptille (koodilistaus 12).

```
  # GET /products/balance/1/1.json
  def balance

    #first is used because we want to return only one object to
json
    @product = Product.joins(:warehouse_entries =>
       [{:warehouse_entry_spots =>
              [{:warehouse_spot =>
                     :warehouse
              }]
       }]
    ).select(
       'products.id,
       products.name,
       sum(warehouse_entry_spots.remaining_spot_quantity) as
balance')
    .where(:id => params[:product_id])
    .where('warehouses.id = ?', params[:warehouse_id])
    .group('products.id')
    .first

    respond_to do |format|
      format.json { render json: @product }
    end
  end
```

Koodilistaus 12. Rails-käsittelijän metodi.

Tuotteen varastosaldokyselyä varten Railsin config/routes.rb-tiedostoon täytyy tehdä reitti, johon tuotteen resurssi vastaa. Tiedostossa esitellään HTTP-pyynnön tyyppi, resurssin polku ja resurssinkäsittelijä ja käsitteljän metodi.

```
# Product
get 'products/balance/:warehouse_id/:product_id' =>
'products#balance'
resources :products
```

`resources :products` -komento luo tuoteresurssille muut REST-arkkitehtuurimallin mukaiset HTTP-pyynnön polut. Rails yhdistää käsittelijän metodit ja HTTP-pyynnöt taulukon 1 määrittelemällä tavalla.

Taulukko 1. HTTP-pyynnöt ja niitä vastaavat käsittelijän metodit.

| HTTP-pyynnön tyyppi | Polku | Käsittelijän metodi | Toimenpide |
|---|---|---|---|
| GET | /products | index | näyttää kaikki tuotteet |
| GET | /products/new | new | palauttaa HTML-formin, josta voi luoda uuden tuotteen |
| POST | /products | create | luo uuden tuotteen tietokantaan HTML-formin perusteella |
| GET | /products/:id | show | näyttää halutun tuotteen tiedot |
| GET | /products/:id/edit | edit | palauttaa HTML-formin tuotteen muokkaamista varten |
| PUT | /products/:id | update | päivittää halutun tuotteen tietokantaan |
| DELETE | /products/:id | destroy | poistaa halutun tuotteen |

## 5.5 Rails ActiveRecord

Rails-sovellyskehykseen ohjelmoitaessa SQL:ää ei tarvitse kirjoittaa, koska Railsistä löytyvä ORM-luokka ActiveRecord on abstraktoinut tietokantatason. Jokaiselle luodulle mallille voidaan Railsissä kutsua eri metodeja, jotka palauttavat tietokannasta löytyvän datan (koodilistaus 13).

```
@products = Product.all
```

Koodilistaus 13. Rails-mallin kutsuminen.

Koodilistauksen 13 lähdekoodi palauttaa kaikki product-oliot, jotka löytyvät tietokannasta.

```
@product = Product.find(params[:product_id])
```

Koodilistaus 14. Rails-mallin kutsuminen.

Koodilistaus 14 lähdekoodi palauttaa yhden product-olion HTTP-pyynnöstä tulevan product_id:n perusteella.

ActiveRecordiin on implementoitu suurin osa tietokantaan liittyvistä ominaisuuksista, kuten esimerkin balance-funktiosta voidaan todeta.

Ohjelmoija voi myös käyttää SQL-kieltä, jos niin haluaa. Jokainen ActiveRecored::Base:sta periytyvä luokka toteuttaa metodin find_by_sql(), jonka sisään voi kirjoittaa SQL-kieltä. Jokaisesta tietokannasta löytyvästä rivistä luodaan sitä vastaavat olio. Ohjelmoija voi myös käyttää metodia select_all(), jolle voi myöskin syöttää SQL-kieltä, mutta se palauttaa hash-taulun löytyneistä tietokantariveistä. Tämä

metodi eroaa find_by_sql-metodista siten, että se ei luo uusia olioita löytyneiden rivien perusteella.

Kaikki ActiveRecordin tekemät tietokantakyselyt tallentuvat palvelimen lokiin SQL-kielenä (koodilistaus 15)

```
Started GET "/products/1" for 127.0.0.1 at 2013-03-31 20:08:06
+0300
Processing by ProductsController#show as HTML
  Parameters: {"id"=>"1"}
  Product Load (72.4ms)  SELECT `products`.* FROM `products`
WHERE `products`.`id` = 1 LIMIT 1
  Rendered products/show.html.erb within layouts/application
(2.4ms)
Completed 200 OK in 104ms (Views: 29.0ms | ActiveRecord: 72.4ms)
```

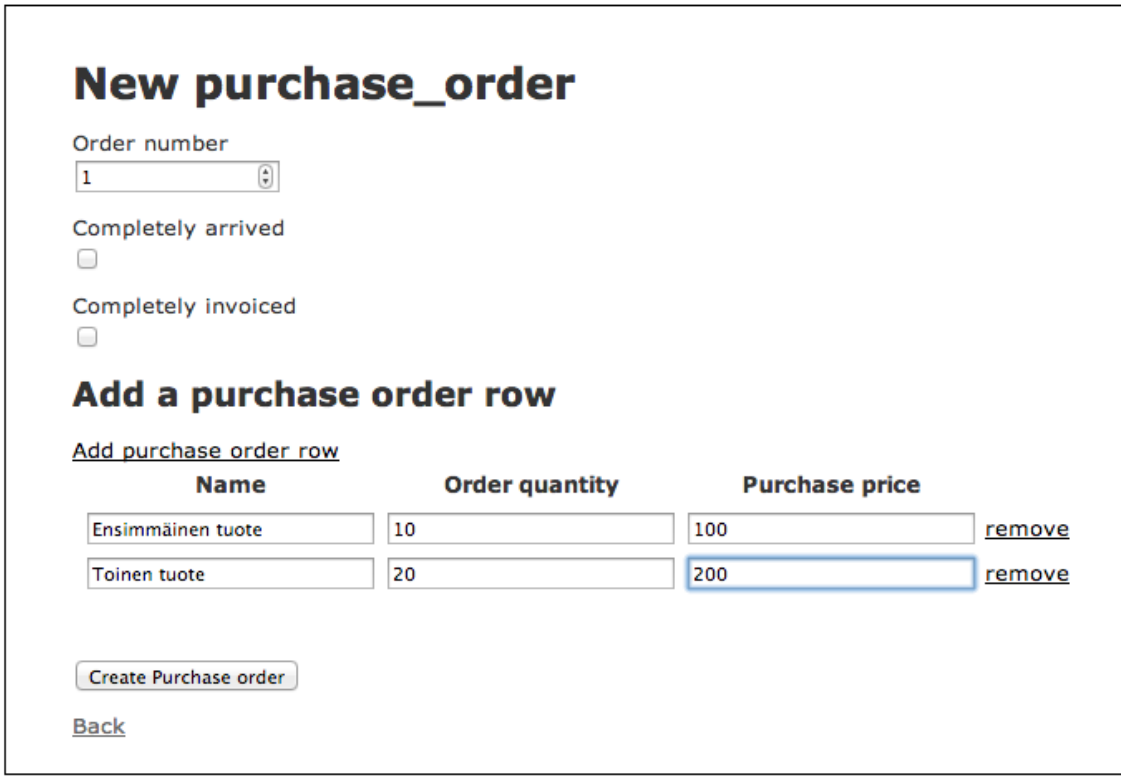
Koodilistaus 15. WEBrick-palvelinloki.

Lokissa näkyy pyydetty resurssi ja sille annetut argumentit. Pyydetyn resurssin ja argumenttien perusteella tehdään tietokantakysely, jonka suoritusaika näkyy kyselyn edessä. Pyynnön kokonaissuoritusaika näkyy viimeisellä rivillä. Nämä tiedot helpottavat kyselyiden testaamista ja optimointia.

# 6 Sovelluksen käyttöesimerkkejä

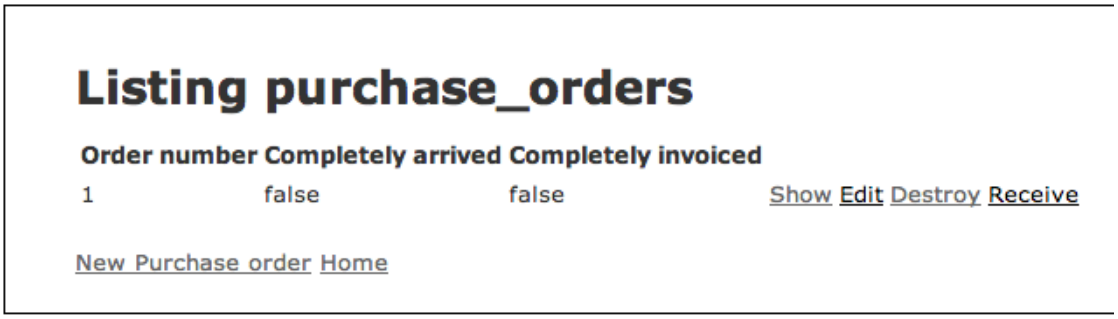
## 6.1 Ostotilaus ja saapuminen

Ostotilauksen ja siihen liittyvän saapumisen tekeminen onnistuu seuraavanlaisesti. Luodaan uusi ostotilaus ja valitaan siihen liittyvät tuotteet (kuva 12). Käyttöliittymästä voi myös antaa ostotilauksen tilausnumeron, jotta tilauksen löytäminen tietokannasta olisi myöhemmin helpompaa. Näkymässä on kentät tilauksen kokonaan saapumista ja kokonaan laskutusta varten. Näiden toiminnallisuus tullaan toteuttamaan myöhemmissä versioissa. Ostotilausriveille syötetään tilattava kappalemäärä ja ostohinta.




Kuva 12. Ostotilauksen tekeminen.

Kun ostotilaus on tehty se voidaan ottaa vastaan receive-linkistä (kuva 13).



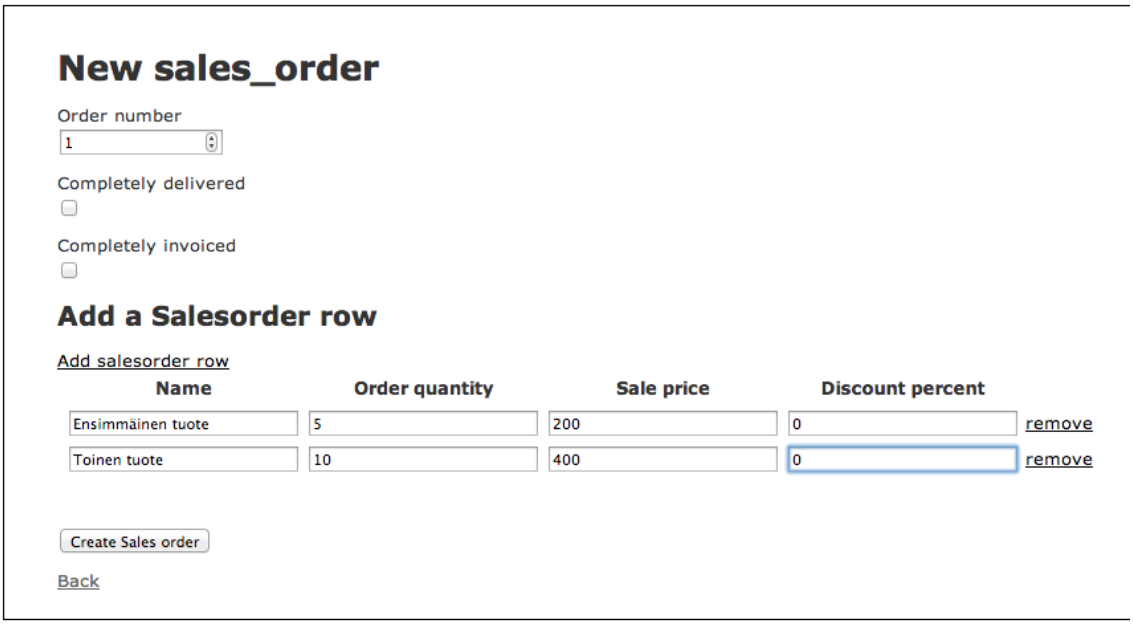

Kuva 13. Tehty ostotilaus ja sen vastaanottaminen.

Ostotilauksessa tilatut tuotteet ja niiden kappalemäärät täyttyvät automaattisesti saapumiselle. Käyttäjän pitää valita varasto, johon tuotteet on vastaanotettu, saapuneet määrät sekä varastopaikat. (kuva 14).


Kuva 14. Saapumisen tekeminen.

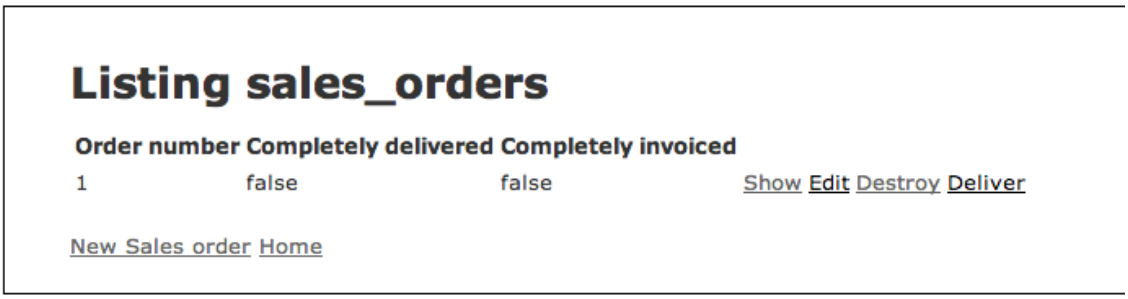
## 6.2 Myyntitilaus ja lähtevä toimitus

Kun varastoon on saapunut tuotteita, niitä voidaan tilata. Myyntitilauksen tekeminen onnistuu myyntitilauksen luomisnäkymästä (kuva 15). Käyttöliittymästä voi myös antaa myyntitilauksen tilausnumeron, jotta tilauksen löytäminen tietokannasta olisi myöhemmin helpompaa. Näkymässä on kentät tilauksen kokonaan toimitusta ja kokonaan laskutusta varten. Näiden toiminnallisuus tullaan toteuttamaan myöhemmissä versioissa. Myyntitilausriveille syötetään tilattava kappalemäärä ja myyntihinta ja alennusprosentti.



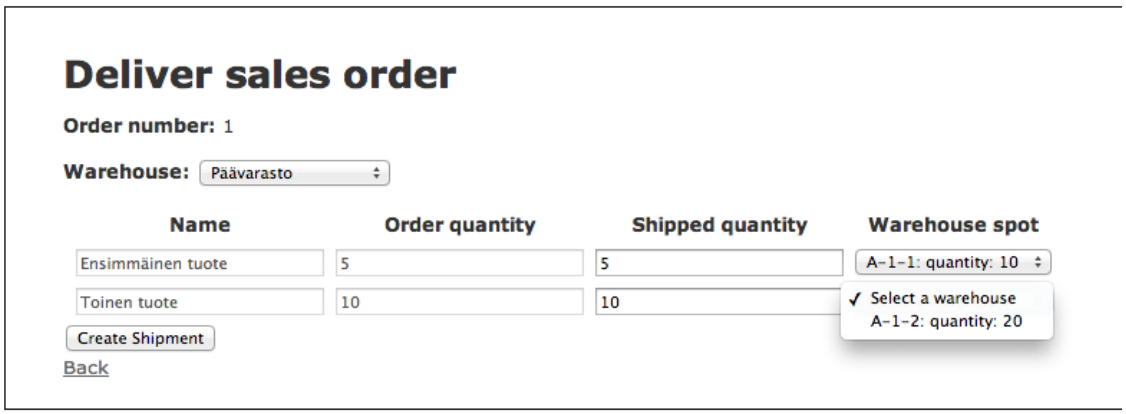

Kuva 15. Myyntitilauksen tekeminen.

Myyntitilaus on tehty ja sen voi toimittaa deliver-linkistä (kuva 16).




Kuva 16. Tehty myyntitilaus ja sen toimittaminen.

Järjestelmä täyttää automaattisesti toimitettavat myyntitilausrivit sekä ehdottaa valitun varaston perusteella varastopaikkoja ja niistä löytyviä kappalemääriä. Käyttäjän pitää syöttää toimitettava määrä (kuva 17).




Kuva 17. Toimituksen tekeminen.

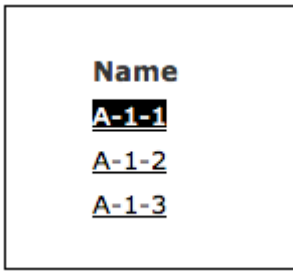
Lähtevä toimitus on tehty. Lähtevän toimituksen näkymässä näkyvät toimitetut tuotteet, kappalemäärät, varasto sekä varastopaikat, joilta tuotteet on kerätty (kuva 18).


Kuva 18. Toimitus tehty.

## 6.3 Varaston inventointi

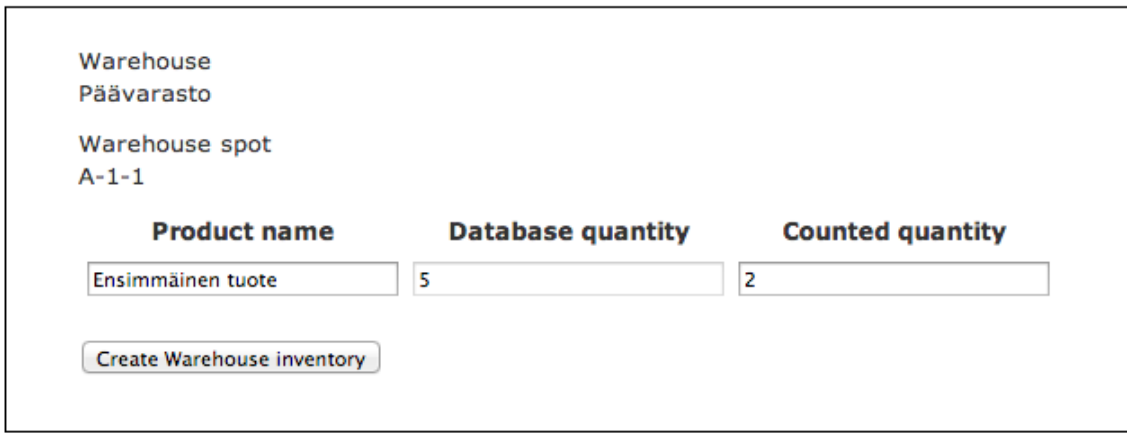
Varastoja voidaan inventoida valitsemalla haluttu varasto (kuva 19) ja varastopaikka (kuva 20).


Kuva 19. Varastoinventoinnin varastonvalinta näkymä.



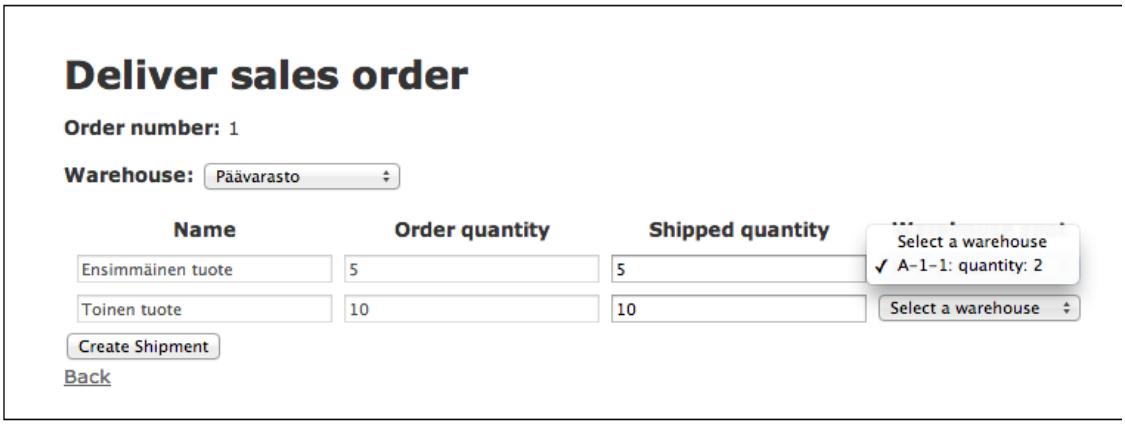

Kuva 20. Varastoinventoinnin varastopaikanvalinta näkymä.

Kun haluttu varasto ja varastopaikka on valittu, järjestelmä näyttää tietokannasta löytyvät tuotteet sekä niiden varastosaldon. Käyttäjän pitää syöttää varastopaikalta löytyvien tuotteiden kappalemäärät (kuva 21).




Kuva 21. Varastopaikan inventointi.

Varastopaikka on inventoitu. Varastosaldon muutos näkyy myyntitilausta tehdessä (kuva 22).




Kuva 22. Ostotilaus varastoinventoinnin jälkeen.

# 7 Jatkokehitys

Varastonhallintasovellus liitetään osaksi toiminnanohjausjärjestelmää. Toiminnanohjausjärjestelmään tullaan ohjelmoimaan myynti- sekä ostoreskontra, asiakashallinta (CRM), käyttäjähallinta, tuotannonohjaus (MES), kirjanpito sekä laskutus.

Toiminnanohjausjärjestelmän ohjelmoinnissa keskitytään erityisesti kirjanpitoon ja laskutukseen, koska nämä ovat yrityksen tuottavuuden kannalta olennaisimpia asioita. Laskutukseen ohjelmoidaan SEPA-standardin mukainen laskutusjärjestelmä. Järjestelmällä näkee yrityksen kaikkien tilien saldot sekä tilitapahtumat realiajassa. Järjestelmällä voi myös maksaa ostolaskuja. Järjestelmän käyttäjän ei siis tarvitse kirjautua verkkopankkiin erikseen maksaakseen laskuja.

Myyntitilausten keräämistä varastosta tehostetaan automatisoimalla keräyslistojen tulostus. Järjestelmä hakee tietokannasta myyntitilauksia toimitusajankohdan perusteella ja tulostaa ne automaattisesti kerääjälle. Kerääjä ei siis voi valita myyntitilauksia vaan järjestelmä valitsee ne kerääjän puolesta.

Myyntitilausten pakkaamista varastolavoille helpotetaan luomalla algoritmi, joka analysoi myyntitilauksen tuotteiden kappalemäärät, koot ja painot. Algoritmi eriyttää painavat tuotteet omille varastolavoilleen ja pitää huolen, että kerääjälle annettun keräyslistan tuotteet mahtuvat kahdelle varastolavalle. Yhden keräyksen maksimi varastolavamäärä on kaksi, koska varastoissa käytettäviin trukkeihin ei mahdu enempää varastolavoja.

Asiakashallintaan ohjemoidaan kommunikointisovellukset. Sovelluksien kautta järjestelmä tallentaa asiakkaille lähetetyt sähköpostit sekä pitää kirjaa soitetuista puheluista. Asiakashallintaan sisältyy myös massasähköpostin lähetys toiminnallisuus. Tämän avulla järjestelmän käyttäjä pystyy seuraamaan asiakkaille lähetettyjen sähköpostien avaamisprosenttia sekä asiakkaiden kiinnostusta eri tarjouksista.

Varastonhallintasovelluksen käyttäjäkokemusta aiotaan parantaa sekä käyttöprosesseja nopeutetaan. Varastosta lähtevien tilauksien hallintaan tullaan toteuttamaan lähtöjen hallinta, jonka avulla lähtevät tilaukset voidaan liittää varastosta lähteviin kuljetusvälineisiin.

Toiminnanohjausjärjestelmään ohjelmoidaan automatiikkaa, joka helpottaa yrityksen päivittäistä kaupankäyntiä. Muun muassa toimitettujen myyntitilausten laskutus, saapuneiden ostotilausten keräyserien tulostus ja varastosta loppuneiden tuotteiden ostilausten lähettäminen automatisoidaan.

# 8 Yhteenveto

Sovellus täyttää ne vaatimukset, jotka alkumäärittelyssä sille asetettiin. Varastonhallinta ei kuitenkaan ole vielä valmis ja kehitystä tullaan jatkamaan. Ensimmäinen versio tarjosi asiakkaalle vain tärkeimmät ominaisuudet.

Sovelluksen ensimmäisen version testaaminen jäi vähäiseksi. Jokaiselle mallille on luotu Railsin automaation kautta yksikkötestit (unit-test), mutta ne ovat täysin riittämättömät eikä niitä ole ajateltu loppuun saakka. Yksikkötestit suunnitellaan erikseen ja niihin panostetaan paljon resursseja, koska testien avulla havaitaan toimiiko järjestelmä oikein.

Työllä saavutettiin asiakkaalle merkittävä helpotus varastonhallintaan jo ensimmäisestä versiosta lähtien aikaisempaan käytäntöön verrattuna. Osto- ja myyntitilauksia voi nyt hoitaa monta eri työntekijää, ja kaikki ovat perillä saapuvista ja lähtevistä tilauksista. Varastotyöntekijöiden työ helpottui huomattavasti, kun järjestelmä osaa kertoa lähtevälle myyntitilaukselle varastosta löytyvät tuotteet sekä niiden varastopaikat. Varastoon saapuvien tuotteiden vastaanotto ei nopeutunut merkittävästi, mutta koska lähtevät ja saapuvat tuotteet käyttävät nyt samaa järjestelmää, nopeutti tämä lähtevien tuotteiden keräämistä huomattavasti.

Järjestelmän ensimmäisen version käyttö on melko kankeaa ja ehkä hieman hidastakin. Automaatioita ei ole tarpeeksi, minkä takia käyttäjät joutuvat vielä tekemään asioita manuaalisesti. Varastonhallinnasta puuttuu myös ominaisuuksia, jotka ovat tärkeitä asiakkaalle. Muun muassa keräyslistojen tulostus pitää toteuttaa mahdollisimman pian. Suoraan myyntitilausten pohjalta tapahtuva tuotteiden kerääminen ei ole kovin tehokasta.

Sovelluksen kehitys tulee jatkumaan. Asiakkaalle välttämättömimmät toiminnallisuudet saatiin toimimaan, sekä asiakas sai asennettua ne tuotantoympäristöön. Asiakas oli tyytyväinen tähänastiseen työhön.

Sovellusta tehdessä huomattiin, että se sopii parhaiten pienille ja keskisuurille yrityksille. Pienen ja keskisuurenyrityksen työntekijä määrä on noin 10—100. Jatkokehityksen myötä sovellus voi olla myös hyödyllinen suuremmillekin yrityksille. Suuremmille yrityksille sovelluksen tietokantarakenne ei ole sovelias sellaisenaan vaan sitä pitää optimoida tai datamäärän kasvaessa dataa pitää tallentaa moneen eri tietokantaan. Olemassaoleva tietokantarakenne on melko hidas varsinkin raportteja ajettaessa. Raporttien tietokantakyselyt saattavat olla lineaarihakuja, mikä tarkoittaa, että tietokantakysely hakee kaikki rivit valituista tietokantatauluista.

## Lähteet


1 Kirjanpidon ABC. (WWW-dokumentti). <http://www.taloushallintoliitto.fi/tilitoimistot/kirjanpidon_abc/>. Luettu 1.11.2012

2 Ruby-ohjelmointikieli.(WWW-dokumentti). <http://www.ruby-lang.org/en/>. Luettu 1.10.2012

3 Matsumoto, Yukihiro ja Flanagan, David. The Ruby programming language. O'Reilly Media, 2008.

4 Interview with David Heinemeier Hansson from Ruby on Rails. (WWW-dokumentti).<http://dev.mysql.com/tech-resources/interviews/david-heinemeier-hansson-rails.html>. Luettu 1.1.2013

5 Make. <http://www.gnu.org/software/make/>. Luettu 1.3.2012

6 jQuery. (WWW-dokumentti). <http://jquery.com/>. Luettu 1.1.2011

7 MariaDB. (WWW-dokumentti). <https://mariadb.org/>. Luettu 1.2.2013

8 Communications of the ACM 13. ACM, 1970

9 Toiminnanohjausjärjestelmä. (WWW-dokumentti). <http://www.nexustek.com/erp-crm/solutions/accounting-erp-software/>. Luettu 1.1.2011

10 YARV. (WWW-dokumentti). <http://ruby.about.com/od/newinruby191/a/YARV.htm>. Luettu 1.4.2013

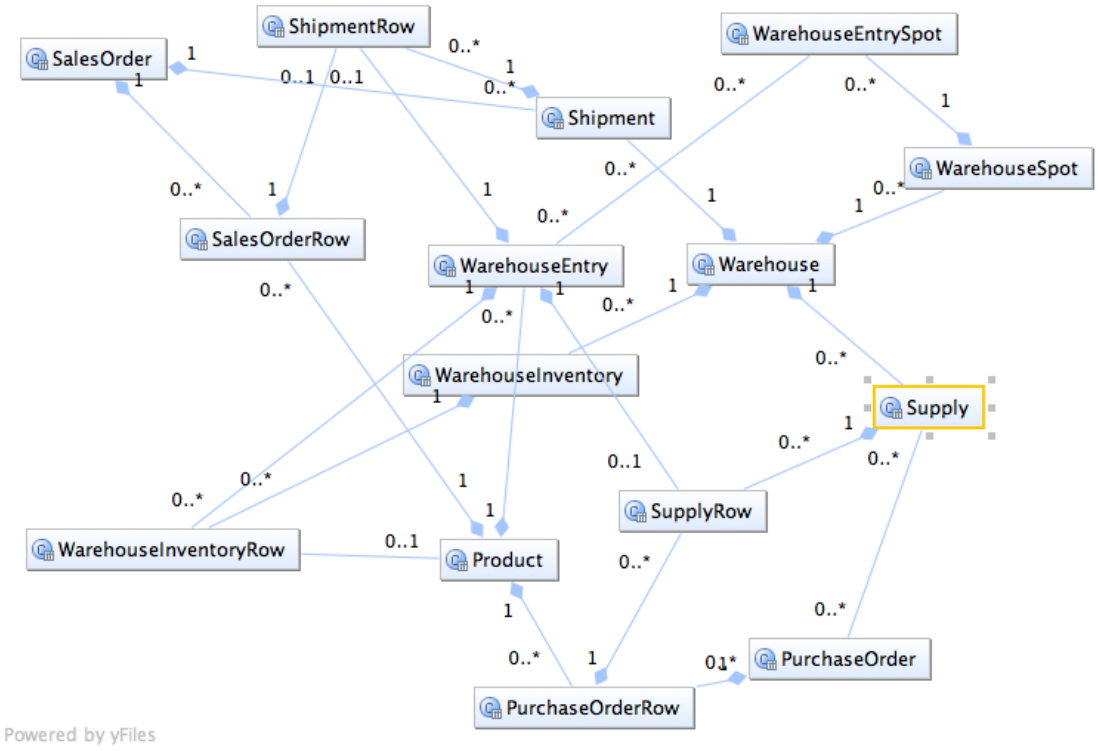
11 Ruby under microscope. (WWW-dokumentti). <http://patshaughnessy.net/ruby-under-a-microscope>. Luettu 1.4.2013

12 HTML. (WWW-dokumentti). <http://www.w3.org/html/> Luettu 1.1.2011

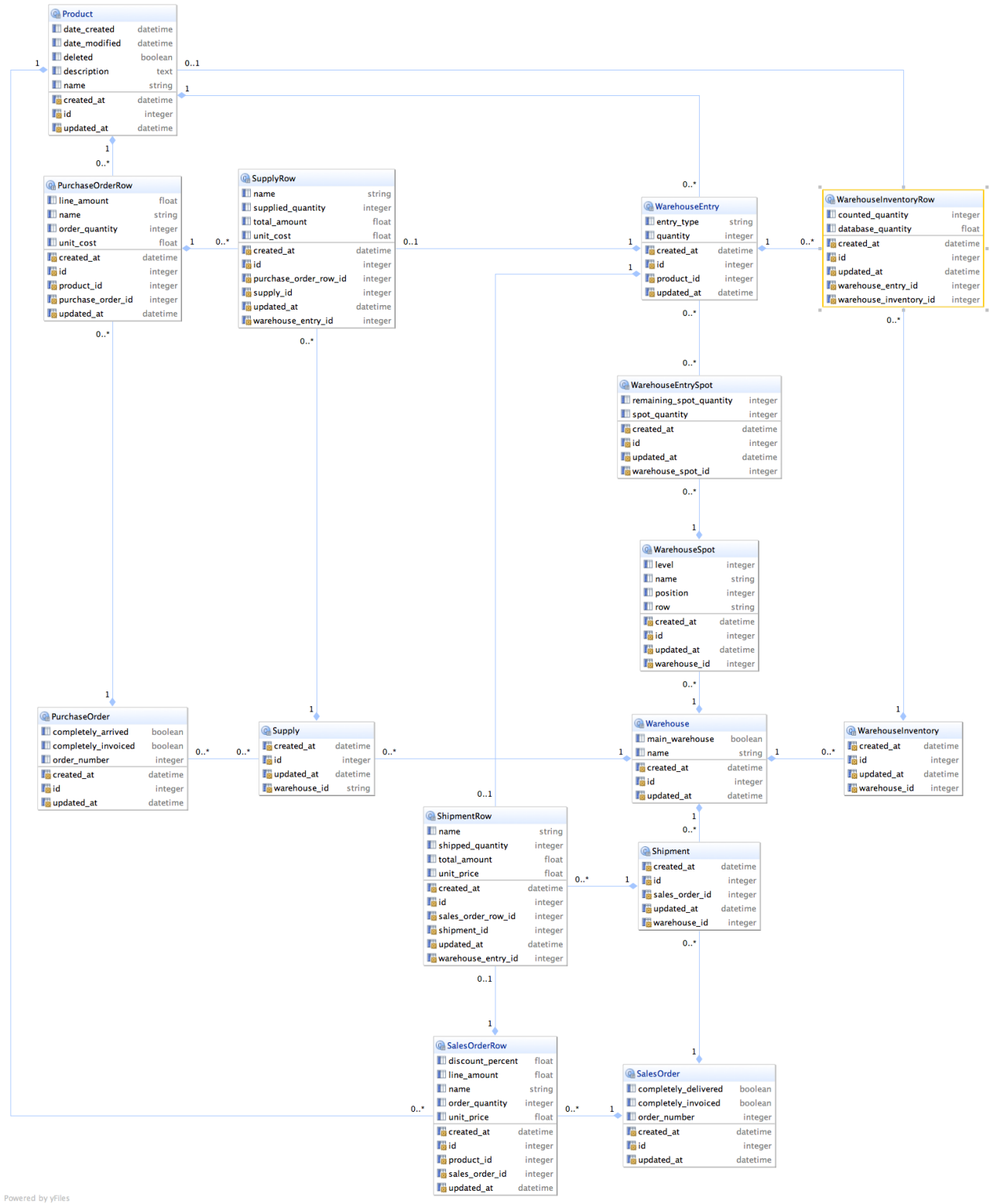
13 CSS. (WWW-dokumentti). <http://www.w3.org/Style/CSS/>. Luettu 1.1.2011

14 Convention over configuration. (WWW-dokumentti). <http://softwareengineering.vazexqi.com/files/pattern.html>. Luettu 1.1.2013

## Liite 1: Malliriippuvuuskaavio



# Liite 2: Relaatiokaavio



**Liite 3: Uuden Rails-sovelluksen luonti komentorivityökalulla**

```
$ rails new wamas
      create
      create  README.rdoc
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      create  app/assets/images/rails.png
      create  app/assets/javascripts/application.js
      create  app/assets/stylesheets/application.css
      create  app/controllers/application_controller.rb
      create  app/helpers/application_helper.rb
      create  app/mailers
      create  app/models
      create  app/views/layouts/application.html.erb
      create  app/mailers/.gitkeep
      create  app/models/.gitkeep
      create  config
      create  config/routes.rb
      create  config/application.rb
      create  config/environment.rb
      create  config/environments
      create  config/environments/development.rb
      create  config/environments/production.rb
      create  config/environments/test.rb
      create  config/initializers
      create  config/initializers/backtrace_silencers.rb
      create  config/initializers/inflections.rb
      create  config/initializers/mime_types.rb
      create  config/initializers/secret_token.rb
      create  config/initializers/session_store.rb
      create  config/initializers/wrap_parameters.rb
      create  config/locales
      create  config/locales/en.yml
      create  config/boot.rb
      create  config/database.yml
      create  db
      create  db/seeds.rb
      create  doc
      create  doc/README_FOR_APP
      create  lib
      create  lib/tasks
      create  lib/tasks/.gitkeep
      create  lib/assets
      create  lib/assets/.gitkeep
      create  log
      create  log/.gitkeep
      create  public
      create  public/404.html
```

```
      create  public/422.html
      create  public/500.html
      create  public/favicon.ico
      create  public/index.html
      create  public/robots.txt
      create  script
      create  script/rails
      create  test/fixtures
      create  test/fixtures/.gitkeep
      create  test/functional
      create  test/functional/.gitkeep
      create  test/integration
      create  test/integration/.gitkeep
      create  test/unit
      create  test/unit/.gitkeep
      create  test/performance/browsing_test.rb
      create  test/test_helper.rb
      create  tmp/cache
      create  tmp/cache/assets
      create  vendor/assets/javascripts
      create  vendor/assets/javascripts/.gitkeep
      create  vendor/assets/stylesheets
      create  vendor/assets/stylesheets/.gitkeep
      create  vendor/plugins
      create  vendor/plugins/.gitkeep
         run  bundle install
Fetching gem metadata from https://rubygems.org/..........
Fetching gem metadata from https://rubygems.org/..
Installing rake (10.0.2)
Installing i18n (0.6.1)
Installing multi_json (1.3.7)
Installing activesupport (3.2.9)
Installing builder (3.0.4)
Installing activemodel (3.2.9)
Installing erubis (2.7.0)
Installing journey (1.0.4)
Installing rack (1.4.1)
Installing rack-cache (1.2)
Installing rack-test (0.6.2)
Installing hike (1.2.1)
Installing tilt (1.3.3)
Installing sprockets (2.2.1)
Installing actionpack (3.2.9)
Installing mime-types (1.19)
Installing polyglot (0.3.3)
Installing treetop (1.4.12)
Installing mail (2.4.4)
Installing actionmailer (3.2.9)
Installing arel (3.0.2)
Installing tzinfo (0.3.35)
Installing activerecord (3.2.9)
Installing activeresource (3.2.9)
Using bundler (1.2.1)
```

```
Installing coffee-script-source (1.4.0)
Installing execjs (1.4.0)
Installing coffee-script (2.2.0)
Installing rack-ssl (1.3.2)
Installing json (1.7.5) with native extensions
Installing rdoc (3.12)
Installing thor (0.16.0)
Installing railties (3.2.9)
Installing coffee-rails (3.2.2)
Installing jquery-rails (2.1.3)
Installing rails (3.2.9)
Installing sass (3.2.3)
Installing sass-rails (3.2.5)
Installing sqlite3 (1.3.6) with native extensions
Installing uglifier (1.3.0)
Your bundle is complete! It was installed into ./vendor/bundle
Post-install message from rdoc:
Depending on your version of ruby, you may need to install ruby
rdoc/ri data:

<= 1.8.6 : unsupported
 = 1.8.7 : gem install rdoc-data; rdoc-data --install
 = 1.9.1 : gem install rdoc-data; rdoc-data --install
>= 1.9.2 : nothing to do! Yay!
```

**Liite 4: Uuden rungon luominen**

```
$ rails generate scaffold Product name:string
date_created:datetime date_modified:datetime deleted:boolean de-
scription:text
      invoke  active_record
      create    db/migrate/20121120183535_create_products.rb
      create    app/models/product.rb
      invoke    test_unit
      create      test/unit/product_test.rb
      create      test/fixtures/products.yml
      invoke  resource_route
       route    resources :products
      invoke  scaffold_controller
      create    app/controllers/products_controller.rb
      invoke    erb
      create      app/views/products
      create      app/views/products/index.html.erb
      create      app/views/products/edit.html.erb
      create      app/views/products/show.html.erb
      create      app/views/products/new.html.erb
      create      app/views/products/_form.html.erb
      invoke    test_unit
      create      test/functional/products_controller_test.rb
      invoke    helper
      create      app/helpers/products_helper.rb
      invoke      test_unit
      create        test/unit/helpers/products_helper_test.rb
      invoke  assets
      invoke    coffee
      create      app/assets/javascripts/products.js.coffee
      invoke    scss
      create      app/assets/stylesheets/products.css.scss
      invoke  scss
      create    app/assets/stylesheets/scaffolds.css.scss
```